EXPLOITING USER COMMENTS FOR WEB APPLICATIONS

XIANGNAN HE

(Bachelor of Software Engineering,

East China Normal University)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2015

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

XIANGNAN HE

12 December 2015

ACKNOWLEDGEMENTS

Doing a PhD is never an easy thing. One must learn to become stronger, both physically and mentally, and most importantly – professionally. I would like to sincerely thank the various people who, throughout my PhD years in which their help lasted, provided me with useful assistance.

First and most important of all, I would like to thank my advisor, A/P Min-Yen Kan for his constant guidance and support all the way. Min has taught me many things, not limited to how to do good research, but how to publicize a research work, how to socialize with other researchers and how to prepare for the future careers. Without Min's generous and unwavering support, I would not have been able to complete this thesis.

I am grateful to the members of my thesis committee, including Prof Wynne Hsu, A/P Lee Wee Sun and A/P Yi Zhang, for their critical reading and assessment of my thesis.

I also appreciate the help from my colleagues including Jun-Ping Ng, Aobo Wang, Jovian Lin, Tao Chen, Muthu Kumar Chandrasekaran, Kazunari Sugiyama and Jinyang Gao, as well as my collaborators Ming Gao, Xiao Chen, Peichu Xie, Yiqun Liu and Yongfeng Zhang for their wisdom, feedback and discussion of my research works.

Although not directly related to this thesis, I am thankful to Google for providing me internship opportunities to experience the life at Google. I would like to thank my internship hosts Bhargav Kanagal, Steffen Rendle, Thomas Sidoti and Sheng Zhang for training me on research and engineering skills. It is really a joy to work with them at Google.

My appreciation also goes out to everybody out there who have supported me in another way throughout my Ph.D. years. These include my girlfriend Yimeng Yang and my friends: Gang Zhao, Bo Yi, Ling Shi, Shangxuan Tian, Chonggang Song, Furong Li, Meng Zhang, Yujing Wei and countless others for bringing love and happiness during the boring Ph.D. life.

Finally, I would like to express my heartful thanks to my family, especially my parents Jing Qin and Guozhu He, for their cultivation and support all the time.

CONTENTS

ABSTRACT

The hallmark of the Web 2.0 era is the incorporation of the user in playing a central role, generating content such as comments, tags, votes and other social actions. As a result, the Web has experienced a renaissance in the form of user-generated content. This characteristic of the new Web has presented new challenges in retrieving, managing and utilizing the great volume of online resources.

In this thesis, we leverage online user comments (or interchangeably "reviews") – with the goal of exploiting the rich signals, such as the presence of temporal dynamics, social influence and user preference – for improving relevant Web applications. We investigate how applications can benefit from leveraging comments, and propose specific techniques for three target applications, namely 1) popularity prediction, 2) item clustering and 3) personalized recommendation. Our investigations into these three applications form the organization of this thesis.

First, we address the item popularity prediction task, by utilizing the temporal dynamics (evidenced by timestamps) found in comments. To alleviate comment sparsity, we additionally model social influence to better predict popularity by coupling it with users' commenting activities. We model comments as a bipartite graph, proposing a graph regularization-based ranking algorithm to encode various ranking hypotheses.

Second, we tackle the task of item clustering, based on the observations that textual comment contents always describe item's properties, and that a user's reviews are often restricted to a limited set of item categories. We leverage this key observation to capture the two extrinsic features from comments – textual words and user IDs – that complement with item's intrinsic features. We formalize the problem as one of multi-view clustering (MVC), proposing a new method that extends the non-negative matrix factorization for MVC and can handle the views (*i.e.,* features) of varying quality well.

To complement the first two works that focused on item-centric applications, our third and final work focuses on a user-centric task: uncovering user's preference from comments for recommendation. We extract aspects (*i.e.,* the specific properties of items) from comment text, analyzing user's preference at a fine-grained aspect level. We devise a transparent and efficient online learning algorithm to provide explainable recommendations for users.

LIST OF TABLES

LIST OF FIGURES

LIST OF PUBLICATIONS

The work in this thesis has been published in the following venues:

- Xiangnan He, Ming Gao, Min-Yen Kan, Yiqun Liu and Kazunari Sugiyama. *Predicting the Popularity of Web 2.0 Items Based on User comments.* In Proceedings of the 37th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'14), pages 233–242, Gold Coast, Australia, July 6–11, 2014.

- Xiangnan He, Min-Yen Kan, Peichu Xie and Xiao Chen. *Comment-based Multi-view Clustering of Web 2.0 Items.* In Proceedings of the 23rd International Conference on World Wide Web (WWW'14), pages 771–782, Seoul, Korea, April 7–11, 2014.

- Xiangnan He, Tao Chen, Min-Yen Kan and Xiao Chen. *TriRank: Review-aware Explainable Recommendation by Modeling Aspects.* In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM'15), pages 1661-1670, Melbourne, Australia, Oct 19–23, 2015.

Note that I am the lead author of all of these works.

Chapter 1

Introduction

The advent of Web 2.0 brought about social media, which evolved traditional, static Web pages through the participation of user-generated content (UGC). Users now not only read content but also participate in a spectrum of social actions: commenting, tagging, voting, forwarding, and so on. As a result, the Web is now more dynamic, complex and larger than before. For example, in YouTube alone, there are over 3 million hours of videos uploaded each week[1], and over 14 million people share, vote or comment on videos each day[2]. Such rich and large amounts of content present new challenges for their retrieval, management, and utilization.

In this context, how can we manage and utilize such Web items most effectively? One key observation is enabled by Web 2.0 itself: the ubiquitous feature of user comments (or interchangeably "reviews"). Most commercial Web 2.0 systems allow users to post comments to express their opinions on the provided items (*e.g.,* articles, images, videos, products *etc.*). For example, Amazon encourages users to comment on the purchased products to solicit user feedback; social media sites like Facebook and Twitter allow users to comment on other people's posts. In addition, some websites that

[1] `https://www.youtube.com/yt/press/statistics.html`

[2] `http://blastmedia.com/2012/10/01/whos-watching-10-mindblowing-youtube-\ statistics`

(a) A video posted by *GoogleTechTalks* **(b) A video posted by *Kahkeshan Sard***

Figure 1.1: An example of user comments drawn from two YouTube videos, motivating our three proposals: 1) timestamps that imply item's future popularity; 2) words that reveal item's category; and 3) content that uncover user's personal preference.

are specifically designed for reviewing items so as to better service users have sprung up in recent decades, like Yelp, Dianping[3] and Douban[4]. User comments are a rich source of information, containing not only the *textual contents* that state users' opinions, but also *timestamps* which record the comment's history, and *usernames* which indicate users' identities for mining their personal preference. Such rich information sources have been shown to benefit a wide range of applications, including information retrieval [107], blog search [95], document summarization [54], image ranking [113] and recommendation system [98].

1.1 Motivation and Challenges

This thesis focuses on exploiting the rich signals in user comments to improve relevant Web applications. Figure 1.1 shows a motivating example, which shows recent user comments drawn from two YouTube videos[5] at a

[3]A Chinese website that has similar functionality with Yelp: `http://www.dianping.com`
[4]A reviewing website for movies/books/musics: `http://www.douban.com`
[5]`https://www.youtube.com/watch?v=1wcX4fEdNUo`
`https://www.youtube.com/watch?v=cd_Fdly3rX8`

particular time. First, from the timestamps of comments, we can see the video on the right garnered more attention than the one on the left at the crawled time, suggesting that video on the right will be more popular in the near future. Second, from the textual content, we can find that the left one is a lecture video about information retrieval, while the right one concerns music. Thus, it is possible to infer the category of an item from user comments alone. Third, by analyzing a user's historical comments on videos, we can uncover his/her personal preferences and interests. For example, by examining the extended comments, we find the user *Matthew Baggott* always comments on e-learning videos, while *Benjamin Rodger* actively watches music videos.

In this thesis, we leverage user comments – such a rich information source – to address the three tasks: predicting items future popularity, clustering items into semantic groups, and generating personalized recommendations for users. The first two tasks mine comments for item-centric applications, while the third task concerns the user-side customization. These three tasks are cornerstones for many applications and we review them in turn:

1.1.1 Popularity Prediction

Predicting the popularity of web content has a wide range of applications, such as online marketing [67], search ranking [36], among others. The primary strategy of the prior work has been to mine the view history of items [2, 106, 119]. However, such solutions are infeasible when item's view histories are not accessible, such as for some external services (which are not the original content provider). Even though many Web 2.0 sites provide a current view count for items, repeated crawling to build and maintain such view histories is expensive, and these solutions also do not allow prediction

for newly crawled items due to insufficient view history.

To address this, we propose to leverage user comments for predicting item's popularity, serving as an alternative solution when view histories are inaccessible. As the timestamps of comments record users' footprints on items as well, it is intuitive to replace the view history with comment history and then adapt the previous view-based solutions [2, 106, 119]. However, a key difficulty with this approach is that commenting activity is much more sparse than viewing: a user viewing an item often does not comment on it; and as such, simply applying existing solutions on comment history is insufficient, especially for less popular items with few comments. As such, this key challenge requires us to exploit the additional popularity signals from comments to combat the sparsity for quality prediction.

1.1.2 Clustering

Clustering has been an effective method to address information overload on the Web in several particular contexts: in automatically organizing web resources for content providers, and in diversifying search results in web document ranking [16]. It has improved retrieval effectiveness for text [142], images [60] and videos [47]. Improved clustering of web resources also helps to automatically generate more meaningful tags [69].

As motivated in Figure 1.1, user comments are well-suited to complement item categorization – 1) the textual content of comments often describes items from the perspective of users, and 2) users themselves are typically interested in a limited categories of items matching their interests, implying that user identities can also be used as clustering features. Aside from comments, items themselves yield intrinsic features (meta information), such as textual description for videos and pixels for images. These different features – two extrinsic features from comments and item's

intrinsic features – carry heterogenous information and can vary vastly in efficacy towards clustering quality. The primary challenge here is how to effectively combine the heterogenous features for improved clustering.

1.1.3 Personalized Recommendation

Recommender systems target at providing interesting content for users, playing an important role in improving user satisfaction and increasing revenue for content providers. Existing research have largely focused on the collaborative filtering techniques [61, 64, 78], which model user preference from the binary user–item relation data, such as ratings. Aside from users' ratings, we find that their attached comments often provide the underlying rationale for their ratings while simultaneously revealing their preferences. Figure 1.2 gives an illustrative example that shows the comments posted by a Yelp user[6]. From her comments, we find that she is more interested in *chicken* and *shrimp*, as she has reviewed restaurants featuring these foods multiple times. As such, if the recommender system can understand that these specific aspects are the ones she is interested in from her comments, it can provide more desirable recommendations to the user.

However, leveraging comments in such way is non-trivial for machines, as these reviews are written by users freestyle, exhibiting noise and irrelevant content. Recent research efforts [8, 32, 51, 80, 92, 136] have largely modeled review words by projecting them into latent topics to combine them using the latent factor model. Although these methods achieve good prediction accuracy, the recommendation process is not transparent and the generated recommendations are not explainable to users, a well-known drawback of the latent factor model [64]. Moreover, these sophisticated methods only provide one-shot recommendation, which is prohibitive in online learning scenarios where new data flows into the system continu-

[6]User ID "*8fTTvS499XCz4oP49kxq8A*".

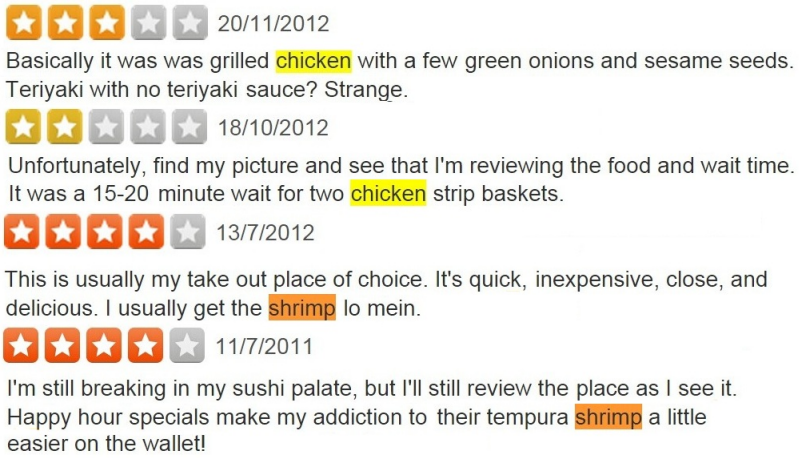


Figure 1.2: Comments posted by a sampled Yelp user.

ously. To improve users' experience and trust, transparency, explainability and efficiency sufficient to enable online learning are critical for practical recommender systems [126].

1.2 Contributions of the Thesis

This thesis makes contributions in review mining, which are closely related with the areas of information retrieval, recommendation system and applied machine learning. They are summarized as follows:

1. **Using comments to predict item future popularity.** To deal with the sparsity in comment-based popularity prediction, we model and utilize social influence. Different from traditional time series-based methods that perform regression on view series, we distinguish the weight of each point (*i.e.,* comment) by considering the social influence of the commented user. We refine the social influence of a user by considering his initial influence (*e.g.,* number of friends), level of activity and the popularity of commented items. Our method works by modeling user comments as a time-aware bipartite graph, on which we propose a novel ranking algorithm – BUIR (*Bipartite User-Item Ranking*) – that captures 1) temporal, 2) social and 3)

current popularity factors for popularity prediction. Extensive experiments on three real-world datasets show our method consistently outperforms traditional methods in several evaluation tasks. This work is presented in Chapter 3.

2. **Using comments to cluster items into semantic groups.** To combine the heterogeneous features of varying quality, we appeal to the technique of *multi-view clustering*, where each feature type represents a view of possibly different clustering utility. We propose a new method, CoNMF (*Co-regularized NMF*), extending the well-known non-negative matrix factorization [70] technique for multi-view clustering by pair-wise co-factorization, which is arguably more effective in digesting noisy views. To further enhance the performance, we devise the initialization (pre-training) and normalization methods for CoNMF. We operate CoNMF in comment-based clustering by modeling the comment words, users, and intrinsic features of items. Experiments on two real-world datasets show CoNMF outperforms state-of-the-art multi-view clustering methods in incorporating comments, validating its advantage in combining views of varying quality. Chapter 4 discusses this work.

3. **Using comments to improve the personalized recommendation for users.** To generate more explainable and transparent recommendations for users, we capture item aspects mentioned within review that models user's preference with more precision. We diverge from using the existing popular approaches [8, 32, 51, 80, 92, 136] that jointly model review words and ratings in latent space, which are opaque to understand. Instead, we first distill textual comments into semantic *aspects*, which represent item's specific properties. Then we model user preference at the aspect level, inferring user's interests on

the aspects by collaborative filtering. We propose a new graph-based method TriRank, which operates transparent and explainable recommendation based on the user–item–aspect tripartite graph. Moreover, TriRank can easily support online learning by instantly personalizing user preference given the new data, which is an attractive property for practical recommender systems while existing solutions only provide one-shot recommendation. Experiments on two public datasets demonstrate TriRank's effectiveness in leveraging comments for enhanced recommendation. Chapter 5 elaborates this contribution.

Chapter 2

Literature Review

User comments, as one of the most common forms of user-generated content, have received substantial research attention in recent decades. We categorize research on user comments as consisting of two paradigms – *comment-targeting* and *comment-exploiting* [108]. Under the first paradigm, the comments themselves form the targets of research task, such as summarizing the sentiment and opinion of comments [52, 89, 147, 152], ranking comments by helpfulness and quality [26, 50, 63, 85, 117], detecting spam comments and spammers [59, 77, 75, 97, 130], among others. The focus of this thesis is on the second paradigm – comment-exploiting, *i.e.* the commented *items* or *users* serve as the targets of research task, while comments are exploited as an additional information source to help the task. For example, comments have been studied to raise the retrieval recall [20, 25, 95, 100, 107, 137], improve the summarization of Web items [29, 53, 54, 103], benefit rating prediction and recommendation [34, 80, 92, 116, 148], help ranking items from user's perspective [93, 113, 133, 144] and so on.

In this chapter, we review works under the comment-exploiting paradigm. We do not seek an exhaustive study on this topic, as there are lots of works of this area. In particular, we first review representative works in exploiting comments for Web applications, and then discuss the related works

on utilizing comments for the three tasks — popularity prediction, item clustering (categorization) and personalized recommendation — which are most related with our proposals. We leave the reviewing of methodologies for each specific task in the related work section of each chapter.

2.1 Comment-exploiting Overview

Hu and Liu [52] first proposed mining user comments as two major tasks: feature identification and orientation prediction, where feature describes the properties of items that the comment talks about and orientation represents the sentiment (*i.e.*, positive, negative or neutral) of the opinions expressed in the comment. Following a similar line, we generalize these two categories introduced by [52] and discuss prior work as belonging to either: 1) descriptive information mining and 2) sentimental information mining, depending on which kind of information within user comments are utilized.

Descriptive Information Mining. The descriptive information latent in user comments have been utilized for many applications, such as search ranking, blog summarization and so on. Mishne *et al.* [95] worked on blog research and found that integrating the textual content of comments improved recall by 5–15%. Later, Potthast *et al.* [107] showed that using comments for cross-media retrieval significantly increased recall by up to 40%, when compared with using titles alone. Similar finding was made by [20] that including comments related social features improve the accuracy of video retrieval. In another application of blog summarization, [54] demonstrated that integrating comments into analysis could also improve the accuracy.

Despite noise in comments is a well-known source of difficulty, the textual content of comments still shows great usefulness for many applications

when properly handled, for example, Fillippova *et al.* [33] improved video classification by incorporating comments with proper noise filtering. Musat *et al.* [98] filtered opinion words by word frequency and improved hotel recommendation. Recently, Yin *et al.* [138] exploited comments to improve the modeling of social tagging systems.

Sentimental Information Mining. The sentimental information latent in user comments can also be leveraged for various applications. Wijaya and Bressan [133] ranked movies based on the sentiment of reviews. Their obtained ranking was highly correlated with the gross income of movies and thus can be used to predict the sales of movies. Later, Zhang *et al.* [144] ranked products on Amazon.com by aggregating opinions mined from customer reviews with the aim of generating a ranked list for each item aspect. Pedro *et al.* [113] extract image features and opinions from comments in order to rank images from an aesthetic perspective. More recently, Zhang *et al.* [148] performed phrase-level sentiment analysis on user comments to improve the accuracy and explainability of recommender system.

Although these works have mined comments for various applications, their focus was exclusively on the textual content of comments. However, usernames (or unique user IDs) and timestamps are complementary and important features that can be extracted from comments, but have been largely ignored in existing work. We believe there is important knowledge latent in the user communities (evidenced by usernames) and timestamps, and that mining these signals can benefit the task of popularity prediction. We thus propose to exploit user comments, especially making sense of the timestamps and usernames information, for popularity prediction. In the next section, we study how the previous works use comments information for the task of popularity prediction.

2.2 Exploiting comments for Popularity Prediction

Works that have considered leveraging comments for predicting item's popularity include [55, 62, 95, 123, 127]. Mishne and Glance [95] first studied the commenting patterns in Web blogs, and then analyzed the relationship between a blog's popularity and the patterns of the comments on it. Their key finding is that the ability of the comments to reflect popularity lessens with less popular items. This lends support to our argument that exploiting just the comment counts alone are insufficient for popularity prediction.

In Kaltenbrunner *et al.*'s work [62] and Tatar *et al.*'s work [123], they considered the popularity prediction problem as predicting number of comments that an item would receive in the future. Specifically, [62] analyzed Slashdot[1] discussion threads, finding that the double log-normal distribution fit post-comment intervals well. Tatar *et al.*'s work [123] treated comments as a general time series and adopted regression methods for count prediction. These two works, however, use the number of comments as the popularity metric, which differs with the more objective measure – view count. We believe that this is a key insight and propose to leverage this property in our own proposal.

Jamali and Rangwala's work [55] on Digg[2] transformed the popularity prediction problem into a classification task. They used user comments as features input to machine learners. Similarly, Tsagkias *et al.* [127] worked on online news, explored prediction as a two-stage classification task: a 0/1 binary classification on whether an article would receive comments, and a second binary classification predicts an article's future comments would be of "low" or "high" volume. However, for these classification-based methods,

[1] `http://slashdot.org/`
[2] `http://digg.com/`

the outputs were too coarse-grained for integration into many applications, such as Web search ranking and online advertising. Moreover, [55] used *Digg-score* as the popularity index and evaluation metric, which was too domain specific.

We argue that one common drawback of the previous works on exploring comments for popularity prediction is that they have solely used comment counts for prediction, while ignoring other signals latent in comments. To amend the sparsity of comment counts, it is important to exploit additional popularity signals for quality prediction.

2.3 Clustering Items Based on User Comments

User comments have been shown to contain useful signals for categorizing and clustering the commented items. Filippova and Hall [33] examined YouTube video categorization. They find that although comments are quite noisy, they do provide useful, complementary and indispensable information for video classification, while the intrinsic features of video title, description and tags are not always indicative of the most relevant category. In a different domain, Li *et al.* [74] cluster blogs, showing that incorporating evidence from the textual content of a blog's comments improves over using the content (*i.e.*, title and body) of the blog alone. Later on, Hsu *et al.* [49] addresses the text of comments, proposing a more comprehensive processing pipeline to de-noise comments. They employ both term normalization and key term extraction before clustering. While these works are both seminal in showing the efficacy of comments in the categorization of items, they only examine the textual content of comments, and ignore the identity of the contributing users, which is a valuable data source for

clustering.

To the best of our knowledge, only Kuzar and Navrat's work [66] on Slovak blog clustering has used the identity of the commenting users. They find that users typically comment on similar blogs, and that such implicit relations produce clusterings that differ from content-based clustering. Crucially they show that a combination of both content- and comment-based analyses yields better overall clustering. However, their combination method is heuristic: they first cluster blogs using only blog content. They then identify the decile of blogs with lowest clustering confidence, and refine their clustering based on the commentator-based clustering.

From the above work, we have strong evidence that comments are useful in clustering Web items. However, previous work has yet to comprehensively utilize all parts of the user comments, focusing primarily on the textual content. To the best of our knowledge, no work has yet to provide a comprehensive study of comment-based clustering, nor provided an effective solution to combine the commenting users' identity, textual content from comments, and item-intrinsic features for clustering.

2.4 Integrating Comments into Recommendation

User comments have been utilized to assist recommender systems in many domains, for movies [32], hotels [98], restaurants [105] and e-commerce [92]. Regardless of domain, we can categorize the approaches based on how reviews are integrated into the recommender: 1) word-based, 2) sentiment-based, and 3) aspect-based methods.

Word-based. These approaches directly factorize the review words into CF. For example, [124, 125] use words to measure similarity, whereas

[51] models each word as a latent vector within the latent factor model. As the original word space is large and sparse, dimension reduction techniques have been adopted. McAuley and Leskovec [92] employs Latent Dirichlet Allocation (LDA) [11] to winnow down the word space, and combines with a latent factor model. Subsequently, [8] has employed Non-negative Matrix Factorization [115] as a replacement of LDA. Others [80, 136] have adopted a full Bayesian treatment to combine topics and latent factors for rating prediction.

Sentiment-based. These approaches utilize the user's explicitly mentioned opinions on items. [102] proposed to fill in the missing ratings with a predicted sentiment score before applying neighbor-based collaborative filtering. [105] built a user–item opinion matrix, where each entry is the aggregated sentiment score of a review, and then applied traditional CF on the opinion matrix. More recently, [32] proposes an integrated graphical model to jointly model the sentiment, aspects and ratings for movie recommendation.

Aspect-based. Our proposal falls into this category. Early work [34] along this line manually annotated six aspects of restaurant domain (*e.g.*, service, ambiance, *etc.*), and classified sentences with respect to these aspects. Their regression method validated the usefulness of aspects for rating prediction. Musat *et al.* [98] built topical profiles of users and items from reviews, and predicted ratings at the topic level. They tested two ways to extract topics, LDA and opinion word frequency, and find the latter produces topics that are more aspect-like. Recently, Zhang *et al.* [148] jointly factorized the traditional user–item rating matrix by inserting aspects, decomposing it into item–aspect and user–aspect matrices, where the aspects are automatically extracted.

Despite the categorization, several hybrid methods have also integrated

aspect and sentiment [32, 34, 148] as they are closely related. These works have jointly modeled aspects, sentiments with ratings to provide better recommendations. In contrast, our work focuses on integrating aspect into collaborative filtering, with the main aim of generating more explainable recommendations. Thus we forgo incorporating sentiment to minimize the reliance on sentiment analysis accuracy. Compared to these review-aware joint latent factor models [32, 80, 92, 148] that are sophisticatedly designed for providing one-shot recommendation, our method explores a graph model to integrate aspects. It has two major advantages: 1) the recommendation process is transparent by explaining its results with respect to aspects, allowing users to customize their recommendations to fit their preferences (*i.e.,* scrutability [126]); 2) it can adapt to new data in real-time by instantly personalizing user preference, easily supporting online learning without retraining.

Chapter 3

Mining User Comments for Popularity Prediction

3.1 Introduction

Modeling and predicting the popularity of web content can benefit many downstream applications such as online marketing [67], cache managing [2], social network modeling [17], search ranking [36], and so on. In this work, we equate predicting *popularity* with the task of predicting future view count, a direct and objective means to assess the hotness of item and reflect users' interest.

Although many existing works have focused on popularity prediction, their primary strategy is to mine the view history of items [2, 106, 119]. However, for some external services which are not content providers, previous solutions are infeasible as they require full access to item's view histories [36]. While many Web 2.0 sites often provide a current view count for items, repeated crawling to build and maintain such view histories is expensive. This method also does not allow prediction for newly crawled items, due to insufficient view history.

To address these challenges faced by external observers, we propose an

alternative approach by exploiting user comments, which are more easily accessible than view counts. In addition to the comment count received in the recent time period – which is a good indicator of future popularity when the comment volume is rich – we further consider the social influence of commented users. This will benefit the prediction for sparse items that have insufficient comment volume to reflect their future popularity. Our method models user comments as a time-aware bipartite graph, on which we propose a graph regularization-based algorithm *Bipartite User-Item Ranking* (BUIR) to rank items by capturing the hypotheses about the future popularity of an item.

3.2 Related Work

In this section, we selectively review representative works on the topic of predicting popularity of online content. For a more comprehensive review, please refer to a recent survey [122].

3.2.1 Popularity Prediction of Online Content

We classify the popularity prediction works into three broad types: statistics-based, classification-based, and model-based approaches.

Statistics-based Prediction. These approaches assume that past popularity is a good predictor of future popularity. Szabo and Huberman [119] first analyzed the popularity growth of YouTube videos and Digg stories, finding a strong correlation between the logarithmically transformed past popularity and current popularity. They proposed a univariate linear model to capture this correlation. Later, Pinto *et al.* [106] extended the univariate model to a multivariate one by incorporating additional historical points and features. Similarly, Lee *et al.* [71] performed survival analysis of two online forums applying Cox proportional hazard regression

to estimate the likelihood that an item will become popular. Radinsky *et al.* [109] proposed several time series prediction methods of user behaviors based on state-space models. All of these techniques require access to the view histories of items, which are difficult for third parties to obtain in practice.

Classification-based Prediction. These approaches transform the popularity prediction problem into a discrete classification task. Different techniques have been utilized, including the k-nearest neighbors [55], decision tree [55, 129], naive Bayes [141]. Various features derived from the textual content, time series, and community structure are distilled as input features for the classifiers. The output of such methods are unfortunately, too coarse-grained for many applications, such as web search ranking and recommendation.

Model-based Prediction. Model-based approaches are difficult to formulate, but often yield more insight and higher accuracy. Yin *et al.* [139] ranked potentially popular items from early votes. They modeled each user's voting behavior as a constrained random process. Recently, Ahmed *et al.* [2] predicted popularity by modeling the temporal evolution of online content. They first split an item's history into time windows, and then generated clusters of items in each window. Based on the clusters, they built a transition graph to predict the most likely cluster for an item in the future.

The above methods, however, only model a user's past behavior or an item's history individually, and do not account for social signals, an important criterion in Web 2.0. Lerman *et al.* [73] analyzed friending actions in Digg as a way of propagating user behavior to influence other users. They modeled user voting behavior as a stochastic model, considering both social influence and website layout to predict story popularity. This work lends evidence that users exert varying levels of influence on others, and

that such social factors need to be taken into account to predict popularity. However, their work is too specific to Digg; parts of their model do not easily transfer to other sites (requiring many internal factors like page view distribution and the number of views per time unit for each page), making the lessons drawn from their study difficult to port to other Web 2.0 sites. For items with longer lifecycles where external events may exert a strong influence, such as unexpected view bursts (*e.g.,* YouTube videos), their approach may not work well.

3.3 Feasibility Study

As there are no previous works using comments as a surrogate to predict the views of items, we first conduct a feasibility study on a YouTube dataset[1] to validate our idea of using comments as a vehicle for popularity prediction.

3.3.1 Correlation of Comments and Views

When a user views an item and feels it interesting, he/she will comment on the item. As such, it is reasonable to conjecture that an item's comment history and view history are highly correlated. Here, we wish to gauge the quality of their correlation to see whether we can use the comment history as a surrogate to predict future views. While prior works [95, 20] have shown that comments do exhibit a strong correlation with views, this is only done for a particular temporal *snapshot* (*i.e.*, on some given day d, how highly correlated are the total cumulative view count with the comment count). To ensure the feasibility of our approach, we need to analyze how the histories of views and comments on individual items evolve over time. To the best of our knowledge, there has not been any studies on such correlation.

[1]The detail of the dataset is described later in Section 3.5 Experiments.

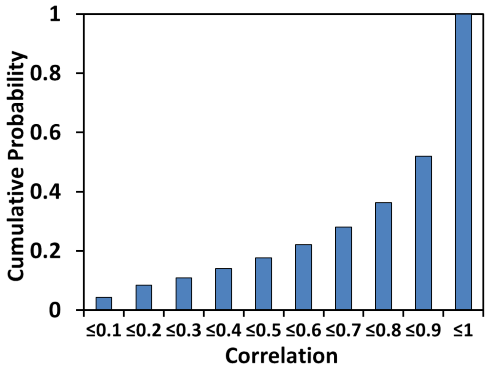


Figure 3.1: CDF of videos with respect to their correlation.

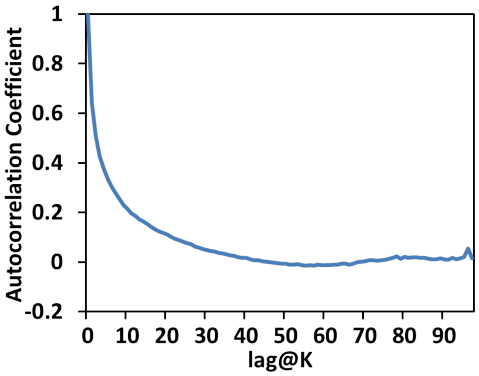


Figure 3.2: Auto-correlation of comment series against lag k.

The historical views of a video form a time series. We first calculate raw counts per time point, then measure the similarity between the comment history and view history using the correlation [18]:

$$cr = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2}}, \tag{3.1}$$

where $x_1, ..., x_n$ and $y_1, ..., y_n$ denote the comment series and the view series, $\overline{x}$ and $\overline{y}$ denote the mean of the two series, respectively.

The mean correlation coefficient for each item is 0.76, with a standard deviation of 0.3. Figure 3.1 shows the cumulative distribution function (CDF) of videos given their correlation. As the figure shows, more than 45% have a correlation greater than 0.9 (strong correlation), and more than 80% have a correlation greater than 0.5 (good correlation). We conclude that *comment history is highly correlated with view history*, which lends supports for our comment-based prediction proposal.

3.3.2 Comment Series Autocorrelation

The strong correlation between the comment history and view history indicates that we can substitute "view" for "comment". Can past comments be used to predict future views, as we propose?

To answer this question, we further perform an autocorrelation analysis of the comment series. The autocorrelation coefficient measures the correlation of a time series with itself over different lags. Given the time series $x_1, ..., x_n$ and a lag k, the autocorrelation coefficient of the series $\{x_i\}$ at lag k is the correlation of series $x_1, ..., x_{n-k}$ and series $x_{k+1}, ..., x_n$. It is usually approximated as follows [18]:

$$acr_k = \frac{\sum_{t=1}^{n-k}(x_t - \overline{x})(x_{t+k} - \overline{x})}{\sqrt{\sum_{t=1}^{n}(x_t - \overline{x})^2}}. \tag{3.2}$$

Figure 3.2 shows the mean autocorrelation of the comment series at different lag k ($0 \leq k \leq 97$). The figure exhibits a short-term correlation characterized by a large value, $acr_1 = 0.64$, followed by a few further coefficients which are successively smaller ($acr_2 = 0.51, acr_3 = 0.43$). Values of acr_k for longer lags ($k \geq 40$) are approximately zero. We thus conclude that *comment histories can reflect future comment in the near-term*, and that *its predictive ability decreases with a larger lag.*

3.4 Proposed Method — BUIR

As most applications seek to determine an item's ranking relative to others, we focus on the relative ranking of items – rather than exact popularity prediction – to reflect their potential popularity in the future (as in [139])[2].

Having shown a strong correlation between the comment and view series, an intuitive solution is to apply any time series prediction approach on the comment series. However, we notice that comments are relatively

[2]There is a similar argument between the task of rating prediction (evaluated by an error-based measure like RMSE) and top-K recommendation (evaluated by a ranking-based measure like NDCG) in the literatures about recommender system [24, 111]. Recent studies on recommendation have shifted the focus from rating prediction to top-K, as most practical recommender systems aim at generating a top list of items for users. Although error rate is a reasonable proxy to measure the prediction quality, improvement of a lower error rate may not translate to a better top ranking [24]. This is also the case for popularity prediction, and we show it in Section 3.5.2

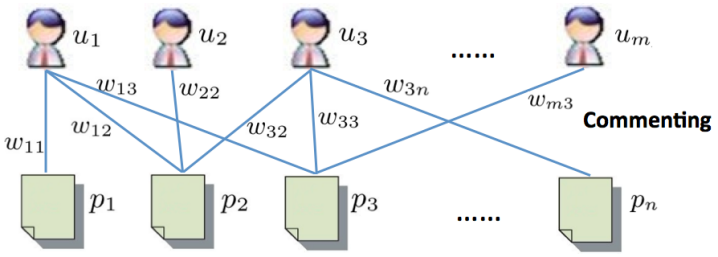


Figure 3.3: Bipartite User-Item Structure.

sparse compared with views, *e.g.,* , many items do not have any comments in a particular time unit. This has an adverse impact on traditional regression methods. We argue that in case of comments sparsity, just using the comment counts is insufficient; it is essential to incorporate other factors for prediction, such as social influence. To account for such latent signals in user comments, we first model user comments as a time-aware bipartite graph, and then predict future popularity based on this graph using a regularization framework [151], which enables the incorporation of multiple factors in a principled manner.

3.4.1 Bipartite User-Item Temporal Graph

Let $G = (U \cup P, E)$ be a bipartite graph, where U and P represent users and items respectively, and the edges E represent comments (Figure 3.3). Each edge carries a weight w, modeling its contribution towards an item's future popularity. As our analysis shows a strong near-term correlation, we assign w based on temporal considerations. We model recent (older) comments as contributing more (less) to an item's future popularity, by assigning edge weight as a monotonically decreasing exponential decay function:

$$w_{ij} = \delta^{a(t_0 - t_{ij})+b}, \tag{3.3}$$

where δ is the decay parameter that controls the rate at which w_{ij} changes with time, t_0 is the ranking time and t_{ij} is the commenting time of user u_i on item p_j. a and b are constants, to be tuned for the particular media

and site. Time units are arbitrary; they can be assigned as minutes, hours, days, weeks or other units, depending on the temporal resolution and the domain of items to rank. If no edge exists between u_i and p_j, then w_{ij} is zero.

3.4.2 Bipartite User-Item Ranking algorithm (BUIR)

We now present our proposed ranking method in the bipartite user-item graph. We describe the hypotheses that form the basis for our regularization function first before presenting our solution.

Hypotheses on Comment-based Prediction

Generally speaking, we have three hypotheses about the future popularity of an item, and wish to incorporate into our model:

H1. Temporal Factor: If an item receives many recent comments, it is more likely to be popular in the next time step (*cf.* our study of YouTube).

H2. Social Influence Factor: If the users commenting on an item are more influential, the item is more likely to receive more views in the future. This is enabled by the Web 2.0 social interfaces that propagate a user's comments to friends and followers. Such social factors have been shown to be useful in popularity prediction and recommendation [73, 90].

H3. Current Popularity Factor: If an item is already popular (*i.e.*, has accumulated a large amount of views), it is likely to garner more views in the future. This is effected by the ranking functions and recommendation interfaces in Web 2.0: the more views an item has, the more likely it will be suggested by the system. This "rich-get-richer" effect has been observed in some Web 2.0 systems [20].

H1 has been studied in our initial analysis of YouTube dataset. We

further validate *H2* and *H3* through experiments in Section 3.5.5.

Regularizing the Hypotheses

We now devise regularizers to capture these three hypotheses. Our goal is to devise a ranking function $f : P \cup U \to \mathbb{R}$, which maps each vertex in G to a real number such that the value reflects the vertex's popularity (for items) or influence (for users).

Capturing H1 and H2. Combining *H1* and *H2* together yields an equivalent formulation: *if an item is reviewed by many influential users recently, it should be given a high score.* We model this through the regularized term $R_1(f)$:

$$R_1(f) = \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{m} w_{ij}\left(\frac{f(p_j)}{\sqrt{d_j^p}} - \frac{f(u_i)}{\sqrt{d_i^u}}\right)^2, \tag{3.4}$$

where w_{ij} is the edge weight defined in Eq. (3.3); n and m denote the number of items and users, respectively; d_j^p and d_i^u are the weighted degrees (*i.e.*, sum of edge weights) of item p_j and user u_i for normalization, respectively.

We now discuss the relationship between $R_1(f)$ and our hypotheses $H1$ and $H2$. First, minimizing $R_1(f)$ forces p_j's normalized score (*i.e.*, $f(p_j)/\sqrt{d_j^p}$) to be similar to the normalized scores of all its connected users. Thus, if p_j is commented on by influential users, its normalized score will be large (as in *H2*). Second, note that the score of p_j is normalized by $\sqrt{d_j^p}$, which is proportional to the degree of p_j. Hence, in constraining the normalized score of p_j to be similar to the scores of its neighbors, $f(p_j)$ is large when the degree of p_j is large (as in *H1*). Therefore, minimizing Eq. (3.4) simultaneously captures both *H1* and *H2*.

It is worth mentioning that the design of sqrt weighted-degree (*i.e.*, $\sqrt{d_j^p}$ and $\sqrt{d_i^u}$) is inspired from the discrete graph regularization frame-

work developed by Zhou [151]. In our scenario, it helps to suppress the highly connect nodes, while other smoothing functions like *log* might also be applicable here.

Capturing H2. We have enforced the social influence of user commenting behaviors on the popularity of items, however, we have not distinguished influential users. Intuitively, if a user has more friends, his behavior is likely to influence more users. Thus, we set a user's initial influence score proportional to the log value of his number of friends:

$$u_i^0 = \frac{\log(1+g_i)}{\sum_{k=1}^{m}\log(1+g_k)}, \tag{3.5}$$

where g_i is user u_i's number of friends at the ranking time. We use add-1 smoothing to address the case where a user has no friends. The choice of the log function is inspired from the TF-IDF scheme, which has been widely used in information retrieval [4]. It is used here for smoothing frequent users that have many friends to avoid they are overweighted. Other functions, such as the exponential decay function g_i^α with $0 < \alpha < 1$, may have the same effect and can also be applied here. We did not further explore these other options since the log functions worked well empirically.

Now we define the regularized term $R_2(f)$ to encode initial user influence:

$$R_2(f) = \sum_{i=1}^{m}(f(u_i) - u_i^0)^2. \tag{3.6}$$

We note that more accurate social influence models do exist (*e.g.*, [5]), but we have purposefully chosen to rely just on the single feature of the number of friends to make our method easily generalizable to a wide range of Web 2.0 systems.

Capturing H3. To capture the potential "rich-get-richer" effect, we

define the initial score of an item as:

$$p_j^0 = \frac{\log v_j}{\sum_{k=1}^{n} \log v_k}, \tag{3.7}$$

where v_j is the total view count of item p_j at the ranking time. Similarly, the corresponding regularizer to capture the current popularity factor of items is defined as:

$$R_3(f) = \sum_{j=1}^{n} (f(p_j) - p_j^0)^2. \tag{3.8}$$

Regularization function. Having defined the regularizer for each hypothesis, we combine them linearly to obtain a final regularization function:

$$Q(f) = \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{m} w_{ij} (\frac{f(p_j)}{\sqrt{d_j^p}} - \frac{f(u_i)}{\sqrt{d_i^u}})^2 + \alpha \sum_{j=1}^{n} (f(p_j) - p_j^0)^2 + \beta \sum_{i=1}^{m} (f(u_i) - u_i^0)^2, \tag{3.9}$$

where the regularization parameters α and β determine the trade-off among these three terms. The first term is a *smoothness* term, that helps to rank items such that high scores are assigned to items that have been recently reviewed by many influential users (*H1* and *H2*). The second and third terms are for *consistency*, that assert that the final rankings should not overly deviate from their initial scores, which encode our hypotheses *H2* and *H3*.

Solving the Regularization

The regularization function $Q(f)$ defined by Eq. (3.9) needs to be solved (minimized) to obtain the final ranking. As two types of variables (p_j and u_i) exist in the function, we can find the solution using alternating optimization. Differentiating $Q(f)$ with respect to p_j and u_i, respectively,

and letting the derivatives be 0, we have:

$$
\begin{aligned}
f(p_j) &= \frac{2\alpha}{1+2\alpha} p_j^0 + \frac{1}{1+2\alpha} \sum_{i=1}^{m} \frac{w_{ij} f(u_i)}{\sqrt{d_j^p}\sqrt{d_i^u}}, \\
f(u_i) &= \frac{2\beta}{1+2\beta} u_i^0 + \frac{1}{1+2\beta} \sum_{j=1}^{n} \frac{w_{ij} f(p_j)}{\sqrt{d_j^p}\sqrt{d_i^u}}.
\end{aligned}
\tag{3.10}
$$

This is the iterative solution of the objective function $Q(f)$. It is guaranteed to find the global minimum, as $Q(f)$ is strictly convex in both the p_j and u_i variables (the Hessian is positive semi-definite). We show the proof in the Appendix. Other standard optimization techniques (*e.g.*, gradient descent) can also be used; alternating optimization has the advantage of quick convergence.

As the updating rules shown in Eq. (3.10) are linear transformations of $f(p_j)$ and $f(u_i)$, they can be equivalently written in matrix form. Let the ranking vectors be $\mathbf{p} = [f(p_j)]_{n\times 1}$ and $\mathbf{u} = [f(u_i)]_{m\times 1}$, and the initial vectors be $\mathbf{p}_0 = [p_j^0]_{n\times 1}$ and $\mathbf{u}_0 = [u_i^0]_{m\times 1}$. Let matrix $\mathbf{S}_w$ be $[\frac{w_{ij}}{\sqrt{d_j^p}\sqrt{d_i^u}}]_{m\times n}$. We then obtain the neat matrix form of Eq. (3.10):

$$
\begin{aligned}
\mathbf{p} &= \frac{1}{1+2\alpha} \mathbf{S}_w^T \mathbf{u} + \frac{2\alpha}{1+2\alpha} \mathbf{p}_0, \\
\mathbf{u} &= \frac{1}{1+2\beta} \mathbf{S}_w \mathbf{p} + \frac{2\beta}{1+2\beta} \mathbf{u}_0.
\end{aligned}
\tag{3.11}
$$

By further reducing Eq. (3.11), we obtain a nice closed-form solution:

$$
\mathbf{p}^* = [(1+2\alpha)\mathbf{I} - \frac{1}{1+2\beta} \mathbf{S}_w^T \mathbf{S}_w]^{-1} \cdot (\frac{2\beta}{1+2\beta} \mathbf{S}_w^T \mathbf{u}_0 + 2\alpha \mathbf{p}_0). \tag{3.12}
$$

Although the closed form can be obtained, in practical cases – especially when there is a large number of items to rank – the iterative solution is preferable, as the matrix to inverse is $n \times n$. In our experiments, the iterative solution usually converges in fewer than 30 iterations, which is sufficiently efficient. Therefore, in subsequent experiments, we implement

the iterative solution of Eq. (3.11) and adopt the name BUIR (*Bipartite User-Item Ranking*) to refer this specific instance.

3.4.3 Time Complexity Analysis

In this subsection, we analyse the time complexity of BUIR using big O notation.

It is easy to show that a direct implementation of the iterative solution in Eq. (3.11) has a $O(mn)$ time complexity, mainly due to the multiplication of $\mathbf{S}_w^T\mathbf{u}$ and $\mathbf{S}_w\mathbf{p}$. However, note that $\mathbf{S}_w$ is typically sparse, as a non-zero entry denotes a comment by a user on an item. A representation of sparse matrix only needs to account for non-zero entries, instead of all mn entries. As such, the whole time cost of BUIR is $O(lc)$, where c denotes the number of comments, and l denotes the number of iterations executed to converge.

If one only aims to rank items without requiring a completed ranking for users, then the time cost can be further reduced through improved implementation. Embedding the update rule of $\mathbf{u}$ into the update rule for $\mathbf{p}$ in Eq. (3.11), we obtain:

$$\mathbf{p} = \frac{\mathbf{S}_w^T\mathbf{S}_w}{(1+2\alpha)(1+2\beta)}\mathbf{p} + \frac{2\beta\mathbf{S}_w^T\mathbf{u}_0 + 2\alpha(1+2\beta)\mathbf{p}_0}{(1+2\alpha)(1+2\beta)}. \tag{3.13}$$

The above can then be solved with simply iterating the update rule Eq. (3.13) until convergence. Note that transition matrix in the first term ($\mathbf{S}_w^T\mathbf{S}_w$) and the entire second term remain unchanged between iterations, thus they can be pre-computed offline. As such, the online ranking reduces to the straightforward power iteration algorithm for computing the stationary distribution of a Markov chain. Without any optimization, the time complexity is $O(ln^2)$. Our experiments on our largest dataset (YouTube) with over 7M comments took 7.4 seconds to complete on a modest commodity desktop (Intel quad-core 3.40GHz CPU and 8GB RAM). And in our Flickr and

Last.fm datasets, BUIR only takes 0.2 seconds to finish ranking. Coupled with previous work [94] that can further accelerate computation, we believe that BUIR can be applied in real-world large-scale online item ranking.

3.4.4 Extensions

Our solution forms a general framework, easily extendible to incorporate other factors beyond what we have described. For new features related to individual comments, such as content and sentiment relevance, we can estimate their weight in contributing to each item's popularity and integrate them into the definition of w_{ij} in Eq. (3.3). For new features related to individual items or users, we can model them within BUIR's bipartite regularization framework (Eq. (3.9)), by adding corresponding regularized terms.

3.5 Experiments

In this section, we evaluate the proposed BUIR method in popularity prediction. We first present the experimental settings. Then, we evaluate the prediction of all items in each dataset (Section 3.5.2). As the overall ranking of all items does not tell the whole story, we then further dissect the results through evaluating on subsets of items, so as to better understand the results. Specifically, we assess the performance on individual queries to show the feasibility of applying our method to Web search ranking (Section 3.5.3), and study the performance over different popularity levels (Section 3.5.4). Finally, we study the proposed hypotheses in Section 3.5.5.

Table 3.1: Statistics of our three Web 2.0 datasets. Avg C:I denotes the average number of comments per item.

Dataset	#Item	#User	#Comment	Avg C:I
YouTube	21,653	3,620,487	7,246,287	334.7
Flickr	26,815	37,690	169,150	6.3
Last.fm	16,284	77,996	530,237	32.6

3.5.1 Experimental Settings

We crawl three real-world datasets from well-known Web 2.0 sites (demographics in Table 3.1) for experimentation.

1. YouTube (21,653 videos): We use ten general queries, drawing from the most popular tags at collection time (9th August 2012), to generate a corpus of videos: "animal", "car", "food", "football", "game", "movie", "music", "nba", "olympic" and "people". We collect the YouTube pages containing the videos using the YouTube API, requesting the top 1,000 videos using three different order-by sorting criteria: ranking by relevance, view count and published time. From this preliminary corpus, we remove (1) duplicate videos, (2) videos with a low number of comments and views (thresholds set to 10 and 20, respectively).

2. Flickr (26,815 images): We follow the same collection method as in the YouTube case, using the same ten queries. We do not apply any frequency filter as this dataset is more sparse than YouTube.

3. Last.fm[3] (16,284 artists): As Last.fm's search API differs from the two other datasets, we collect this dataset by obtaining data about artists: obtaining at most 100 similar artists for each of the top 1,000 most popular artists. For the query-specific evaluation, we query on the top 10 tags that describe a music style: "classical", "country", "electronic", "folk", "hip-

[3]In lieu of view count, Last.fm provides a "scrobble" count, which is the number of times Last.fm users listen to a track by the target artist. This differs from the view count of an artist's page, but we argue more indicative of an artist's popularity. For convenience, we use "view count" to refer to scrobble count in Last.fm.

hop", "indie", "jazz", "metal", "pop" and "rock". We assign each artist to the single tag that is used most often to describe the artist by Last.fm users.

We choose the three datasets for ease of evaluation, as these all provide item view count to crawl. Our datasets are crawled on two different dates: for graph construction (t_0) and for obtaining ground truth (GT) for evaluation (t_3), which is 3 days after t_0. As we have observed that ephemeral trends are important to capture, we specifically aim to evaluate short-term prediction and chose 3 days as the target period to evaluate. The initial crawl t_0 for YouTube, Flickr and Last.fm is on 9th August 2012, 3rd September 2012 and 24th October 2012, respectively. For items in Flickr and Last.fm, we crawl the view count, the number of friends and the list of comments on the two dates. For YouTube, due to its privacy policy, we cannot obtain a user's number of friends, so we set the initial score for users uniformly. As older items may have accumulated many past comments but which would not significantly contribute in BUIR, we discard comments older than five months before t_0 for efficiency. If an item is commented by the same user multiple times, we only keep the most recent comment when calculating the edge weight. This also helps to avoid problems when users have tangential conversations via comments.

Evaluation Metrics

To assess the predicted ranking with the ground truth ranking, we employ ranking correlation in the standard form of the Spearman coefficient [4]. It measures the agreement between two rankings and ranges from -1 to 1, where 1 (-1) means a perfect agreement (disagreement) between the two rankings.

While the Spearman coefficient is indicative of the agreement between two rankings, it does not reflect the importance of getting the top ranks cor-

rect, which are crucial for many applications such as web search ranking. To address this, in the query-specific evaluation we further adopt normalized discounted cumulative gain (nDCG) [56], which rewards relevant results in top ranks highly than those ranked lower. The two evaluation metrics are detailed in [43].

Baselines

We mainly compare with statistics-based approaches, since the outputs of classification-based methods [55, 141] are too coarse-grained for ranking items, and the model-based approaches [139, 73] rely on system-specific designs that are not easily transferable to our datasets.

1. **View Count (VC)**: Rank based on the current view count of items. This corresponds to our belief in Hypothesis *H3* alone.

2. **Comment Count in the Past (CCP)**: Rank based on the number of comments received in the 3-day period prior to t_0 (*i.e.*, t_{-2} to t_0), corresponding to our Hypothesis *H1*.

3. **Comment Count in the Future (CCF)**: Rank based on the number of new comments received in the three days after t_0 (t_1 to t_3). This is an oracular method with access to future comments.

4. **Multivariate Linear model (ML) [106]**: We implement this method on the comment series of the 30 days prior to t_0, aggregating comment counts into 3-day windows, each contributing a feature, for a total of 10 features. This is the state-of-the-art statistical method for predicting the popularity of Web content.

5. **PageRank (PR) [101]**: Our temporal user-item graph is bipartite, which could cause the random walk to become periodic and non-stationary [96]. To work around this, we use the standard method to set a uniform self-transition weight $w_{ii} = 1$ for all nodes, and then convert the weight matrix to a probabilistic one for use with PageRank. For the damp-

Table 3.2: Spearman coeff. (%) of overall evaluation.

Method	YouTube	Flickr	Last.fm
VC	73.39	58.42	67.31
CCP	83.35	59.43	67.21
CCF	84.53	59.41	67.20
ML [106]	78.24	58.00	38.09
PR [101]	80.72	28.15	10.24
BUIR	**87.72**	**64.60**	**70.43**

ing factor, we vary its value from 0 to 1 with step size 0.05. Experimental results show consistently good performance when the damping factor is in the range 0.1 to 0.9; we set it to 0.85 as suggested in [101].

In our BUIR solution, there are two sets of parameters to be specified: 1) ones for assigning edge weights, and 2) ones for the regularization. Edge weights are assigned intuitively: for the time unit of YouTube and Last.fm, as comments are rich and reflective of popularity, we set it to 1 day; for Flickr, we find that the comments are posted less frequently, thus we set it to 3 days; for the time decay function in Eq. (3.3), we empirically set $\delta = 0.85$, $a = 1$, and $b = 0$ for all datasets. As for the regularization parameters α and β in Eq. (3.11), we randomly held out 10% of the dataset as development for parameter tuning. We use grid search to set the best parameters on the development portion, and then evaluate all methods on the remaining 90% test portion.

Note that although the first two baselines are heuristic and simple, they do produce reasonable results for short-term popularity prediction, thus forming competitive baselines (see [109]). For all methods, if items receive the same score, we break ties by ranking based on their current view count.

3.5.2 Overall Evaluation

Table 3.2 shows that BUIR achieves the highest fidelity in ranking items of the test datasets, among all methods. Further experimentation of 10-fold cross validation shows that BUIR obtains very consistent performance, significantly outperforming all other methods ($p < 0.01$, via one-sample paired t-test). BUIR is followed by CCF and CCP, where the difference between CCP and CCF are insignificant. VC also obtains a good performance in general, indicating the effectiveness of *H3*. PageRank (PR) performs poorly for Flickr and Last.fm, indicating that just the centrality of an item in the user-item temporal graph is insufficient for prediction. We also used BUIR's initial vector $\mathbf{p}_0$ and $\mathbf{u}_0$ as the personalized vector of PageRank, which also results in poor performance. This lends evidence that separately handling the two vertex types (users and items) in the bipartite graph is important.

It is surprising that the state-of-the-art ML approach underperforms CCP, as ML leverages more information: comments in the recent 30 days compared with CCP's access to only three days. There are two possible reasons for this: 1) short-term prediction, and 2) ML's optimization criterion. As the prediction task is a short-term one, the most recent data carries the most signal – "What happened yesterday will happen tomorrow." Radinsky *et al.* [109] concurs with this observation, showing that in the short-term prediction of query and URL clicks, considering only the last value of the time series generally outperforms other regression methods, such as using power weighting function and linear weighting function. This also indicates the effectiveness and competitiveness of simple baselines in near-term popularity prediction. The second cause may stem from ML's use of minimizing the mean Relative Squared Error (mRSE) [106]) as its optimization criterion. We note that using mRSE as the optimization metric may favor evaluations on items with a small number of current views,

as the relative popularity growth to learn are larger compared to items with a large current views[4]. As a result, the parameters learned may not be meaningful: we find that optimized weights are sometimes non-sensical (*i.e.*, negative) and that the weights for recent time units can be smaller than the earlier ones, also contradicting intuition. We also find that when we decrease the number of features to learn, performance increases. Thus, although ML does provide a better estimation of future popularity than CCP in terms of mRSE, we believe this criterion does not fit well with the goal of relative ranking. This also highlights the difference between the task of predicting the exact popularity and ranking items by the predicted popularity. Although the (exact) popularity prediction problem is more challenging compared with the relative ranking problem, we believe that the ranking problem is more suited for applications where the ordering (and not the exact numeric quantity) is important: such as search ranking, recommendation and online advertising.

It is worth noting that correlation levels dramatically differ in each dataset. YouTube shows the highest correlation while Flickr is the lowest. This indicates that comments in YouTube are generally richer and thus better reflect trending and popularity growth. Flickr users, as a whole, are less active than YouTube users (as can be seen from the comment statistics in Table 3.1). More specifically, many items do not receive sufficient comments to reflect their future popularity; some items even do not receive any comment within our 5-months window. In these cases, *H1* does not hold, which leads to the degraded performance of comment-based prediction methods.

Let us dissect the ranking lists to gain additional insight. In Last.fm, we notice that BUIR incorrectly ranks two items very high, while their GT

[4]The results of tiered popularity evaluation (Section 3.5.4) reflect this: ML performs better on less popular items in general.

ranks are low. Looking into the data, we find that the two abnormal items are two well-known artists – Lady Gaga and Madonna – ranked 4th and 7th, while their GT rank is 170th and 178th, respectively. After observing the comments, we find that the two artists receive many recent comments, but do not receive a proportional play count. Many comments are about two artists as a persona or just express praise, rather than their music. In Flickr, a similar phenomenon occurs with a few images that are ranked high but have low GT ranks. One[5] has 1,891 comments but only 4,115 views; the other[6] has 1,276 comments but only 3,299 views. Examining the details, we find that many users leave comments for participating in Flickr group activities (*"Good work! I like it!! This photo definitely deserves a Bronze Trophy! FLICKR BRONZE TROPHY GROUP Post"*), which is the cause for the excessive ratio of comments to views. In both the Last.fm and Flickr cases, the items are ranked incorrectly as the comments are not reflective of their intrinsic popularity.

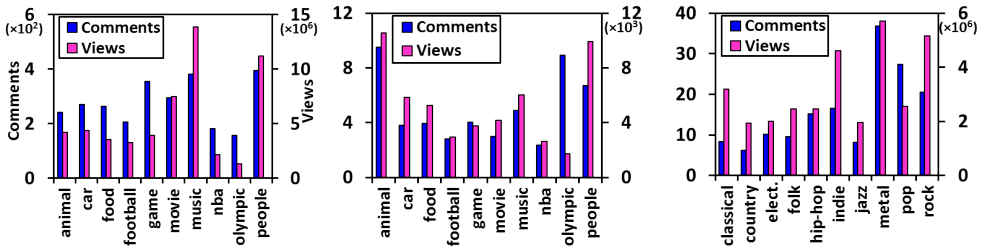
From our two case studies, we can see that if an item is experiencing a burst and the burst is reflected in comments, BUIR successfully ranks it high. However, in the case of items receiving a disproportionally high number of comments to views, disobeying *H1*, BUIR is misled into making incorrect rankings.

3.5.3 Query-Specific Evaluation

We also evaluated prediction quality on a per-query basis to test BUIR's variability for specific queries and its feasibility for use with Web search ranking.

On our datasets, per query, BUIR needs to rank between 500 to $3,500$ items. Figure 3.4 shows the average number of comments and views of

[5] `http://www.flickr.com/photos/jabitxu/7402395070/`
[6] `http://www.flickr.com/photos/jabitxu/6967289760/`



(a) YouTube **(b) Flickr** **(c) Last.fm**

Figure 3.4: Mean of comment# and view# of the ten queries.

Table 3.3: Query-specific evaluation by Spearman coefficient.

Metric	**Spearman coefficient (%)**		
Method	**YouTube**	**Flickr**	**Last.fm**
VC	71.98±14.14	46.72±7.82	67.86±5.76
CCP	82.41± 2.50	48.06±7.90	66.97±4.70
CCF	**83.42±2.7***	48.12±7.80	67.27±4.45
ML [106]	76.95± 5.50	50.00±6.50	39.15±4.04
PR [101]	79.66± 4.72	27.80±14.87	9.22 ±11.66
BUIR	**85.98±5.92***	**55.22± 6.10***	**70.42±4.43***

"*" denotes the statistical significance for $p < 0.05$

items for each query, which highlights the variability in comment and view count between queries; it is not necessary that items with many views have a corresponding number of comments, or vice versa (seen the case of query "pop" and "rock" of Last.fm).

Table 3.3 and 3.4 show the average performance over all queries, evaluated by Spearman Coefficient and nDCG@10, respectively. We also perform one-sample paired t-test (p-value = 0.05) to assess statistical significance. Supporting our previous results of overall evaluation in Table 3.2, BUIR performs the best in all datasets. Specifically, as judged by the Spearman coefficient, BUIR outperforms all baselines except the case of CCF in YouTube, where they are statistically comparable. Surprisingly, for nDCG@10, VC achieves comparable performance (the same significance level) with BUIR in all datasets. As nDCG@10 only evaluates the ranking of the top 10 positions, of which are all popular items, we hypothesize that the current view count is a good indicator of popular items. This motivates

Table 3.4: Query-specific evaluation by nDCG@10.

Metric	nDCG@10 (%)		
Method	**YouTube**	**Flickr**	**Last.fm**
VC	**64.70±22.23***	**67.19±15.75***	**90.25±4.96***
CCP	46.66±29.89	61.35±18.56	82.52±10.85
CCF	**73.04±16.97***	56.94±25.73	78.57±12.83
ML	27.85±30.76	50.74±18.64	74.30±11.15
PR	61.10±21.92	54.53±22.62	81.16±10.07
BUIR	**76.13±12.29***	**74.19±15.70***	**88.19±4.68***

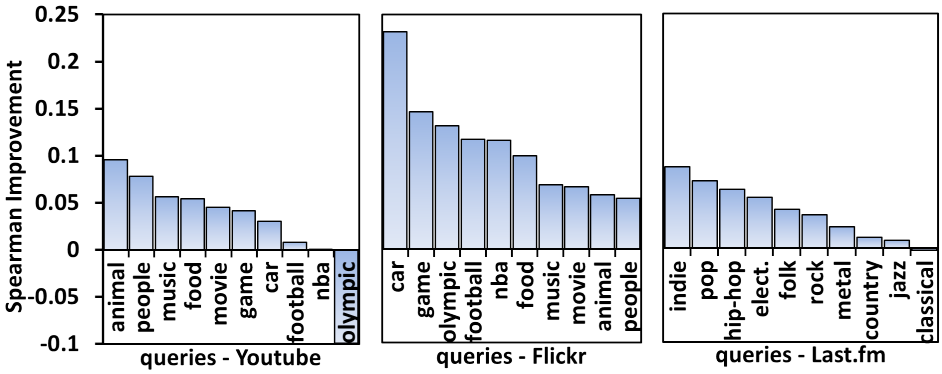


Figure 3.5: Improvement in Spearman coefficient between BUIR and the best baselines of query-specific evaluation.

the need to analyze prediction at other popularity levels (detailed later in Section 3.5.4).

We further examine the performance for each query. Figure 3.5 shows the percentage of improvement in Spearman coefficient between BUIR and the best baselines (CCF, ML, and VC for YouTube, Flickr, and Last.fm, respectively). As can be seen, BUIR bests the baselines compared in all cases with the exception of "olympic" in YouTube and "classical" in Last.fm. We investigate the cause for these performance exceptions.

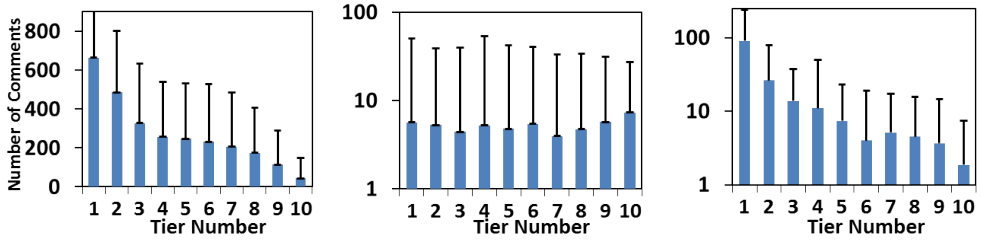
For "olympic", CCF and CCP show a significant improvement over other methods (0.80 for CCF and CCP; 0.72 and 0.34 for BUIR and VC, respectively). The YouTube dataset is crawled on 9th August 2012, during the London Olympic Games. Many collected videos of the query "olympic" from YouTube are indeed about the London Olympic Games. These videos

are rather new, such that they have not accumulated enough view count to reflect their popularity (*cf.* Figure 3.4's view statistics). However, the recent comments are more reflective, as users are actively commenting on the events. From the user comments, we observe that users watch videos largely according to their interests or perhaps their country's medaling in an event. In this case, *H2* (Social Influence Factor) does not strictly hold. Hence, our method does not give the better result. For such new items, we postulate that performance may be improved with a more fine-grained time unit for BUIR. Changing the time granularity to an hourly basis, BUIR's performance improves (from 0.72 to 0.76), although still underperforming CCP and CCF. This lends tentative support to our idea, but which needs further investigation in future work.

For the results of "classical" in Last.fm, VC obtains the higheset Spearman coefficient (0.781), followed by BUIR (0.780) and CCF (0.765). The query "classical" reflects a wide range of classic musicians, such as Frdric Chopin and The Beatles. Such items have existed for a long time, and have already accumulated many views and reached a steady state in attracting views. In these cases, current view count (VC) reflects their future popularity well.

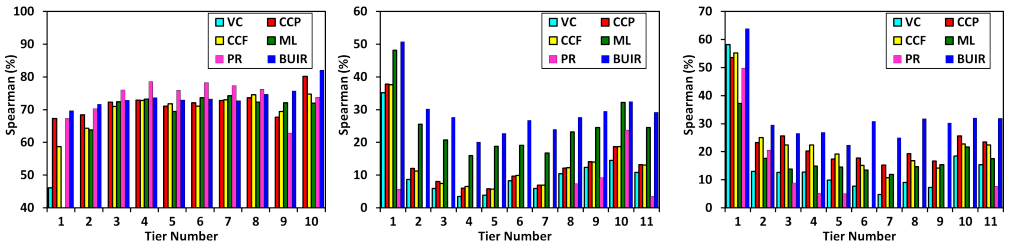
3.5.4 Tiered Popularity Evaluation

While BUIR performs well overall, does it perform consistently on items of different popularity? To answer this question, we study the prediction quality over different popularity levels. We first sort items by descending view count at t_0 and then split into ten equal-sized subsets: Tier-1 (most popular) to Tier-10 (least popular). We report the results for ranking correlation (note that as each tier accounts for a popularity range, nDCG is already considered).



(a) YouTube **(b) Flickr** **(c) Last.fm**

Figure 3.6: Comment statistics (mean and standard deviation) for items in the ten popularity tiers.



(a) YouTube **(b) Flickr** **(c) Last.fm**

Figure 3.7: Results of the tiered popularity evaluation for the three datasets.

Figure 3.6 reports the comment statistics (mean and standard deviation) of the ten tiers. Both the YouTube and Last.fm datasets show the same trend: the average number of comments decreases when moving to higher (less popular) tiers, while all tiers in Flickr do not show much difference. This is because Flickr users largely refrain from making comments compared to YouTube and Last.fm users. As a result, popular items with high view count do not necessarily mean that they will have a high number of comments. From the comment statistics, we can see that the items in high tiers are less popular items with a low number of comments in general.

Figures 3.7 shows the performance broken down by tier. In general, we observe the same trends over all three datasets. Firstly, BUIR consistently performs well and the improvements over the comment-based baselines (CCP and CCF) are more noticeable for higher tiers, corresponding to less popular items. Secondly, current view count (VC) performs well for low tiers while suffers significantly for high tiers, and is worse than CCF

and CCP. As VC ranks well for most popular items (*cf.* nDCG@10 of query-specific evaluation, in Table 3.4), we conclude that the current view count is a good predictor for popular items, but not for less popular items. Furthermore, we also note CCF does not always outperform CCP, although CCF utilizes the future knowledge. This indicates the limitation of simply using the comment count for popularity prediction, and motivates the necessity of mining more signals from user comments for prediction.

For Flickr, BUIR improves over CCP and CCF significantly in all tiers; while in the Last.fm case, BUIR shows slight improvement in lower tiers (less than 5), which represent more popular items. To be precise, the average improvement over CCF in Tiers 1–5 is 5.0%, while in Tiers 6–10, the improvement is 12.1%. We note that the average number of comments in Tier 1–5 is 30.0, while in Tier 6–10 is only 4.3. This indicates that for items in the top tiers (which can be said to have already accumulated sufficient comments), taking social influence into account may not capture much additional signal. Conversely, this highlights social influence as a good signal for prediction of less popular items, earlier given as *H2*.

To conclude the above three sets of experiments, we recap the key findings to predict popularity based on user comments:

1. For popular items which have already accumulated many views, the current view count predicts future popularity well.

2. For items with sufficient number of comments, the recent comments are a good predictor for future popularity.

3. For the bulk of less popular items, neither the current views nor recent comments is sufficient for quality prediction; it is important to incorporate more signals, such as social factors.

4. Most importantly, our proposed BUIR method realizes the most effective and consistent prediction performance, by accounting for temporal,

Table 3.5: Spearman coefficient of overall prediction and performance decrease of different parameter settings.

Setting	YouTube	Flickr	Last.fm
$\alpha = 0$	81.01 (-7.7%)	52.99 (-18.0%)	56.45 (-19.9%)
$\beta = 0$	64.05 (-27.0%)	62.68 (-3.0%)	68.36 (-2.9%)
$\alpha, \beta = 0$	51.24 (-41.6%)	53.77 (-16.8%)	47.22 (-33.0%)

social and current popularity factors.

3.5.5 Hypotheses Study

In this final subsection, we wish to validate the necessity for modeling all three comment-based hypotheses in BUIR. As *H1* is intuitive and has been studied in Section 3.3, we concern ourselves primarily with the *H2* (social influence) and the *H3* (current popularity) factors.

In BUIR, there are two regularization parameters, α and β, which determine the weight of *H3* and part of *H2* (social influence factor captured by users' initial score) in prediction. Table 3.5 shows the prediction performance when regularization parameters are set to 0 (to be clear, a "0" setting nullifies the corresponding factor). As can be seen, when either α or β is set to 0, BUIR suffers and does not predict well; when both α and β are zeroed, the performance further decreases. These results provide additional support to validate our hypotheses *H3* and *H2*. As such, we conclude that every factor captured in BUIR — *H1*, *H2* and *H3* — is necessary for high-quality popularity prediction based on user comments.

3.6 Conclusion

In this chapter, we investigate how to leverage user comments for predicting the popularity of Web 2.0 items. Two signals from comments are identified for this task – temporal and social influence factors – together

with item's current popularity factor are considered. We introduce a new ranking algorithm, Bipartite User-Item Ranking (BUIR), that realizes the three ranking factors under a graph regularization framework. Our BUIR algorithm is generic in ranking vertices of bipartite graphs [44], such that it can be directly deployed for other applications, such as mining terms and definitions in scientific articles [58], and queries and URLs in search engine [30]. Experiments on three different Web 2.0 media — YouTube, Flickr and Last.fm — show the effectiveness of our proposed method on several evaluation tasks. Detailed analysis reveals that the factors individually only predicts well for some subset of items, while combining all under the proposed BUIR methodology yields the highest quality predictions.

Chapter 4

Mining User Comments for Item Clustering

4.1 Introduction

In this chapter, we explore the central theme of how to best process user comments and employ them to cluster Web 2.0 items. This research is timely, as recent work [33, 49] have shown that comments do contain useful signals in discriminating the categories of items, while very few work has studied how to utilize comments to assist item clustering.

As items themselves yield intrinsic features, how to integrate the two extrinsic data sources derived from comments (here, the textual content and the commenting users) is an important consideration. A solution might simply build a unified feature space comprising of the features from all three data sources, such that any standard clustering algorithm can then be applied. However, as the three data sources are generated heterogeneously and may vary drastically in clustering quality, a simple combination method may not achieve optimal performance (our study in Section 4.3 verifies this). As such, the key challenge in comment-based clustering is how to meaningfully combine the multiple evidences for clustering. This

challenge can be addressed by *multi-view clustering*, where each data source represents a view of possibly different utility.

In this work, we present a method that extends the NMF (non-negative matrix factorization [70]) for multi-view clustering. NMF factorizes the data matrix in an easily interpretable way and has shown superior performance in document clustering [135]. While substantial research has been conducted on NMF, studies where NMF is used for multi-view clustering are limited. To address this gap, we propose a CoNMF (Co-regularized NMF) framework, which jointly factorizes multiple views by pair-wise co-factorization. We offer two instantiations – pair-wise CoNMF and cluster-wise CoNMF, and derive the learning algorithms. To further enhance the clustering performance, we devise the initialization (pre-training) and normalization methods for CoNMF. We apply CoNMF on comment-based clustering, demonstrating its effectiveness in combining views of vary quality.

4.2 Related Work

This section reviews the literatures on multi-view clustering, which represents a collection of methods of which our specific proposal of CoNMF is an instance.

4.2.1 Multi-View Clustering Techniques

Work on multi-view clustering can be grouped into three categories – early, intermediate and late integration – based on when the information from the single views are integrated for clustering.

Early Integration. In these approaches, multiple views are first integrated into a unified view, and then input to any standard clustering algorithm. In [28], de Sa fuses the two views' data into a bipartite graph, apply-

ing spectral clustering [99] on the result. In [121], Tang *et al.* project the data matrices from different views into a shared latent space via a proposed linked matrix factorization. Other representative work include [10, 19], which project the multi-view data into a low-dimensional subspace through Canonical Correlation Analysis (CCA). K-means or spectral clustering is then applied to the projected subspace.

Late Integration. In these approaches, each view is clustered individually, and then the results are merged to reach a consensus. Bo *et al.* [86] assume that the optimal clustering should be close to the clustering of all views as much as possible. Bruno *et al.* [14] treat the optimal clustering as hidden factors to generate the clustering of the different views, and then adopt PLSA [48] to solve the problem. Greene *et al.* [38] first concatenate the cluster membership of different views to a unified matrix, and then perform NMF on the unified matrix to obtain the final clustering.

Intermediate Integration. In these approaches, multiple views are fused during the clustering process. Kumar *et al.* [65] propose a co-regularization framework to extend spectral clustering for multi-view clustering. Wang *et al.* [131] propose a mutual reinforcement clustering approach for multi-view interrelated data objects. Their basic idea is to iteratively propagate the clustering results of one view to all its related views. Lu *et al.* [87] cluster webpages from content, social tags and users by a variant of k-means, which calculates the centroid of a cluster based on the features from multiple views. Ramage *et al.* [110] propose Multi-Multinomial LDA, which extends LDA [11] by assuming the latent factors of each single view are generated by a shared distribution. They show superior performance over k-means on clustering webpages from content words and social tags. In addition, similar LDA-based multi-view approach [21] has been proposed to co-cluster images and texts in a soft way.

Our proposed method CoNMF directly extends NMF for multi-view clustering, and is an instance of intermediate integration. It is most similar in spirit to [3, 83]. Akata and Thurau [3] propose to jointly factorize multiple data matrices (views) through a shared coefficient matrix (the W matrix in NMF). This is a hard constraint which may be too strict in some scenarios. Additionally, their method is provably equivalent to early integration, where one first concatenates all views into a unified matrix, and subsequently applies NMF. Recently, Liu *et al.* [83] propose MultiNMF, which regularizes the coefficient matrices learned from different views towards a common consensus for clustering. In their work, a key challenge to address is how to make the coefficient matrix of different views comparable. They employ the L_1 norm on the whole data matrix, and then enforce the same L_1 norm constraint on the coefficient matrix during factorization. We find two weaknesses of their solution in practice. First, when the length of vectors varies greatly across views, the resulting proposed L_1 norm on the whole matrix is biased towards longer vectors[1]. However, their solution integrates the normalization constraint into the optimization framework, making their technique specific to L_1 norm and difficult to extend to other normalization strategies. Second, when the clustering quality of the component views varies greatly, the learned consensus can underperform a single good view, as the poor quality views negatively affect the consensus. Though one can manually tune weights to decrease the effect of noisy views, this parameter tuning process of unsupervised learning is non-trivial.

We address both issues of MultiNMF in our CoNMF method. We co-regularize on each pair of views, which is more robust to the presence of noisy views. This addresses the second issue. For the first issue, we embed

[1]Vector length denotes the number of features derived from an item. Section 4.3 and 4.5.3 demonstrates the impact of normalization on clustering.

the normalization into the optimization process, which enables us to adopt any normalization strategy on the coefficient matrices, effectively offsetting the influence of vector length in multi-view clustering. In addition to our CoNMF solution that resolves the robustness of MutliNMF to noisy views, a later work by Yin *et al.* [140] also targeted at the same problem by injecting the feature selection with a subspace learning method.

Beyond the matrix factorization (MF) method, another more generic paradigm is the tensor factorization (TF), since the a matrix can be seen as a special 2-order tensor. Thus, another solution for multi-view modelling is using the tensor method [23, 120]. However, TF methods for multi-view clustering are still under-explored, and they might suffer from high computational complexity and low interpretability. So in this work, we focus on MF methods, leaving the exploration of tensor methods as future work. It is worth mentioning that our proposed pair-wise factorization method can be seen as a PITF tensor decomposition model [112], which is a special case of the Tucker decomposition.

4.3 Preliminary Study

In this section, we conduct an initial study on a Last.fm dataset[2] to motivate our approach and illustrate the challenges.

We utilize the k-means clustering algorithm [104] for our study. K-means is a widely used, intuitive and efficient clustering algorithm based on the vector space model (VSM).

We want to answer the following questions with our study:

Q1. How do the three views differ in their ability to discriminate different categories of items? Do the views based on user comments help?

[2]The detail of the dataset is described later in Section 4.5 Experiments.

Table 4.1: K-means performance with different settings.

Metric	**Accuracy (%)**			**F_1 (%)**		
View	**Des.**	**Com.**	**Usr.**	**Des.**	**Com.**	**Usr.**
1. Basic	11.8	9.3	8.4	7.5	10.1	9.8
2. Filtered	15.3	9.4	8.6	10.9	10.3	9.8
3. L_1	15.2	19.0	7.9	11.0	13.9	9.9
4. L_1-whole	14.5	9.7	8.5	10.8	10.4	9.8
5. L_2 (count)	15.9	26.9	34.5	10.7	17.6	15.2
6. L_2 (tf)	16.8	25.9	34.7	10.6	17.1	15.3
7. L_2 (tf×idf)	23.5	30.1	34.5	14.5	16.8	14.7
8. Combined	**40.1**			**24.2**		

Q2. How should we preprocess comments to reduce noise and improve clustering efficiency?

Q3. In the VSM, how should each vector be normalized? How should the individual features for each view be weighted?

Q4. How should we combine the three views optimally? Will the resultant combined view yield better clustering?

We run k-means 20 times with random initialization and report the average performance in Table 4.1 when run with different settings described next. The column names "Des.", "Com." and "Usr." represent the item-intrinsic description view, and the two comment-based views (comment words view and users view), respectively. In answering the above questions, we work our way from the basic k-means to answering the issues of noise filtering, normalization, term weighting and view combination, to yield a worthy baseline for comparison.

Basic Feature Space (Row 1). To get a base result, we first build a plain VSM for each view: each item is represented as a row vector. The raw counts of the words or usernames are used as the vector elements. Then, we run k-means on each view's feature space, yielding the perfor-

Table 4.2: Dimensionality of each view, for the original and reduced feature space.

View	Des.	Com.	Usr.
Original	99,405	2,244,330	455,457
Reduced	14,076(−85%)	31,172(−98%)	131,353(−71%)

mance reported in Row 1. The clustering quality is poor, bettering random assignment (accuracy / F_1 of about 6.6% / 5.0%) by a small margin.

Filtering Noisy Features (Row 2). As our textual features are known to be noisy, and the feature space is large, we consider how to filter noise to improve performance. For the two text-based views (the comment words and description views), we first retain only English words, then remove common stop words and conflate the words to stemmed form, using the NLTK toolkit [9]. For the users view, we retain users who had commented on more than 2 items, as users that only comment on few items may not be strong signals for clustering. Table 4.2 shows the dimensionality of the original and reduced feature spaces, where we see a drastic reduction, which aids clustering efficiency. This filtered space's yields improved performance on the description view, while performance on the users and comment words views are unchanged. As such, we take the filtered features as the basis for the remainder of this initial study.

Normalization (Rows 3–5). As normalization influences clustering performance, we assess the impact of different normalization strategies. Item-based L_2 norm, where each item vector is scaled to a unit length, is a widely used scheme for k-means, resulting in Spherical k-means [31]. The item-based L_1 norm yields a unit sum for each vector, which has a probabilistic explanation where feature values represent its probability of occurring in the item, is also often used. In [83], the authors propose using L_1 norm on the whole data matrix (which we denote as L_1-whole), meaning that each entry in the matrix is divided by the sum of all entries.

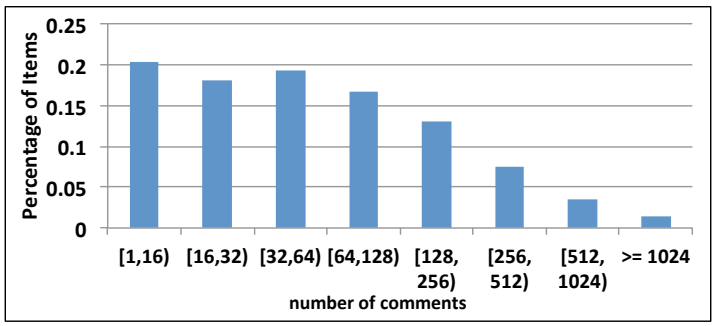


Figure 4.1: Comment distribution of items in the Last.fm dataset.

This results in the elements in the entire data matrix summing to unity, which has the probabilistic explanation where each entry denotes the joint probability of the feature and item.

Rows 3–5 show the results of applying these three normalization strategies. While the results for the description view remain largely unchanged, the comment words and users view are improved, with the L_2 norm outperforming both L_1 and L_1-whole significantly. For the description view, we find that the item's description is contributed by Last.fm's editorial staff and is of a controlled length. As such, the vector length does not vary much across items and normalization has little effect. In contrast, the vector length for the two comment-based views depends on the number of comments on the item, which varies greatly. As shown in Figure 4.1, although most items ($\sim 95\%$) receive less than 512 comments, these items are almost evenly distributed in different intervals. In such a case, normalizing by L_1-whole will still bias towards frequently commented items, while an item-based L_2 norm is more effective in offsetting the influence of vector length for clustering. In the following, we use the item-based L_2 norm. In other experiments where we substituted NMF for k-means, we reach the same conclusion.

Term weighting (Rows 5–7). Feature weighting also influences the clustering process. In information retrieval, weighting based on term frequency and inverse document frequency (tf×idf) are common. We follow

the standards in [4] to implement three common weighting schemes, whose results are shown in Rows 5–7: raw term count (count), term frequency (tf, log of raw term count) and tf×idf. Note that we first weigh the features, before normalizing the vectors with the L_2 norm.

For the two text-based views (description and comment words view), tf×idf performs significantly better than tf and count, while for the users view, all three weighting schemes perform comparably. In the following, we thus use tf×idf for the two text-based views, while using raw term counts for the users view.

Combined view (Row 8). Having benchmarked the clustering performance using the views individually, we assess whether there is benefit in combining the views together using a simple early integration approach. We first normalize each view, and then concatenate all views using the same weight. Formally, let the row vector of an item be $\mathbf{v}_d$, $\mathbf{v}_c$ and $\mathbf{v}_u$ for the three views respectively. Then the combined vector is $\mathbf{v} = [\sqrt{\frac{1}{3}}\mathbf{v}_d, \sqrt{\frac{1}{3}}\mathbf{v}_c, \sqrt{\frac{1}{3}}\mathbf{v}_u]$.

Row 8 shows that such a simple integration performs well, significantly outperforms all of the individual views on both metrics (p-value < 0.01). This results indicates that combining the views is advantageous. Further experiments where we tried different linear weightings of the three views did not further improve performance.

Our preliminary study has benchmarked k-means performance on the clustering of Last.fm artists (items) into genres (categories). We saw that with proper filtering, normalization and feature weighting, the individual views can generate useful clusters and start to answer the four questions posed at the beginning of this section. A key outcome of the study is that the users view (*i.e.*, identity of commenting users) is useful, but potentially overlooked in previous research.

Concluding this preliminary study, we see that early integration by

combining all three views into a single view yields improved clustering performance, answering the second half of Q4. But as the views differ in nature and in innate clustering quality, we suspect that a more principled method of integration may yield even better results. The remainder of this chapter describes our approach to find a convincing framework for answering Q4.

4.4 Proposed Method — Co-regularized NMF

Our solution in finding a principled method to combine views adopts the non-negative matrix factorization (NMF) technique. We first propose the general CoNMF framework to combine multiple views for joint factorization, and then refine two paradigms of the framework – pair-wise CoNMF and cluster-wise CoNMF. As an additional contribution, we further devise a novel k-means based method to facilitate clustering, which can be seen as the pre-training step for CoNMF. The time complexity of our method is discussed in the end.

4.4.1 CoNMF Framework

The hypothesis behind multi-view clustering is that different views should admit the same underlying clustering of the data. Formally, given n_v views denoting as $\{V^{(1)}, ..., V^{(n_v)}\}$, each view is factorized as $V^{(s)} \approx W^{(s)}H^{(s)}$, where $W^{(s)}$ are with same dimension $m \times K$ for all views, while $H^{(s)}$ are of dimension $K \times n^{(s)}$, differing per view; all matrices have the non-negative constraint.

In our CoNMF approach (overview in Algorithm 1), we implement this constraint by coupling the factorization of the views through co-regularization.

Algorithm 1: Co-regularized NMF (CoNMF)

Input: Non-negative matrices $\{V^{(s)}\}$, parameters $\{\lambda_s\}$, parameters $\{\lambda_{st}\}$ and number of clusters K;
Output: Coefficient matrices $\{W^{(s)}\}$ and basis matrices $\{H^{(s)}\}$;
Normalize each view $V^{(s)}$ such that $||V_{i\cdot}^{(s)}|| = 1$;
Initialize matrices $\{W^{(s)}\}$ and $\{H^{(s)}\}$ (Section 4.4.4);
while *Objective function does not converge* ***and*** *Number of iterations* $\leq$ *Threshold* **do**
  for *each s from 1 to* n_v **do**
    Normalize $W^{(s)}$ and $H^{(s)}$;
    Update $W^{(s)}$ and $H^{(s)}$ using **either**
      Eq. (4.8) (Pair-wise CoNMF; *cf* Section 4.4.2) **or**
      Eq. (4.12) (Cluster-wise CoNMF; *cf* Section 4.4.3);
  end
end
return $\{W^{(s)}\}$ *and* $\{H^{(s)}\}$

Generally speaking, the objective function of CoNMF is formulated as:

$$J = \sum_{s=1}^{n_v} \lambda_s ||V^{(s)} - W^{(s)} H^{(s)}|| + R, \quad s.t. \quad W^{(s)} \geq 0, H^{(s)} \geq 0, \tag{4.1}$$

where λ_s are the parameters to combine the factorization of different views and R is the co-regularization function that enforces similarity constraints on multiple views. CoNMF is a general framework as different regularization schemes and similarity measures can be used to implement the co-regularization function R.

4.4.2 Pair-wise CoNMF

To implement the hypothesis of multi-view clustering, an intuitive method is to regularize the coefficient matrices of the different views towards a common consensus, which is then used for clustering. This is the cornerstone of MultiNMF [83] (consensus-based co-regularization). However, a key weakness of this approach is that it fares well only when views are largely homogeneous and of roughly the same quality. In real world applications, different views may be generated heterogeneously and may vary drastically in quality. This is the case that we observe in our comment-

based clustering settings. In the MultiNMF approach, the model's constraints enforce a rigid common consensus that forces views with higher clustering utility to be degraded by ones with lower utility, which may lead to poorer performance.

Pair-wise CoNMF relaxes MultiNMF's constraints, instead of imposing similarity constraints on each pair of views. Through the pair-wise co-regularization, we expect that the coefficient matrices learned from two views can complement with each other during the factorization process. It should thus yield a better latent space and be more effective for clustering.

Intuitively, the co-regularization function of pair-wise CoNMF is defined as:

$$R_1 = \sum_{s=1}^{n_v} \sum_{t=1}^{n_v} \lambda_{st} ||W^{(s)} - W^{(t)}|| = \sum_{s,t} \lambda_{st} ||W^{(s)} - W^{(t)}||, \tag{4.2}$$

where λ_{st} is the parameter to denote the weight of the similarity constraint on $W^{(s)}$ and $W^{(t)}$. Substituting R in Eq. (4.1) with R_1, we obtain the objective function:

$$J_1 = \sum_{s=1}^{n_v} \lambda_s ||V^{(s)} - W^{(s)} H^{(s)}|| + \sum_{s,t} \lambda_{st} ||W^{(s)} - W^{(t)}||, s.t. \quad W^{(s)} \geq 0, H^{(s)} \geq 0. \tag{4.3}$$

We then minimize the objective function to get the solution.

Optimization

Similar to the known solution for NMF, we can adopt alternating optimization to minimize the objective function. The optimization works as follows: (1) fix the value of $W^{(s)}$ while minimizing J_1 over $H^{(s)}$; then (2) fix the value of $H^{(s)}$ while minimizing J_1 over $W^{(s)}$. We iteratively execute these two steps until convergence, or until a set number of iterations is exceeded.

The objective function J_1 can be re-written as:

$$\begin{aligned} J_1 = & \sum_{s=1}^{n_v} \lambda_s Tr(V^{(s)T}V^{(s)} - 2V^{(s)T}W^{(s)}H^{(s)} \\ & + H^{(s)T}W^{(s)T}W^{(s)}H^{(s)}) \\ & + \sum_{s,t} \lambda_{st} Tr(W^{(s)T}W^{(s)} - 2W^{(s)T}W^{(t)} + W^{(t)T}W^{(t)}), \end{aligned} \tag{4.4}$$

where $Tr(\cdot)$ denotes the trace function. Here, $||A|| = Tr(A^TA)$ and $Tr(AB) = Tr(BA)$ are used in the derivation. To enforce the non-negativity constraints, we need to incorporate Lagrange multipliers. Let $\alpha^{(s)}$ and $\beta^{(s)}$ be the Lagrange matrices for constraint $W^{(s)} \geq 0$ and $H^{(s)} \geq 0$, respectively. The Lagrange L_1 is:

$$L_1 = J_1 + \sum_{s=1}^{n_v} Tr(\alpha^{(s)}W^{(s)T}) + Tr(\beta^{(s)}H^{(s)T}). \tag{4.5}$$

Then, the derivatives of L_1 with respect to $W^{(s)}$ and $H^{(s)}$ are:

$$\begin{aligned} \frac{\partial L_1}{\partial W^{(s)}} =& \lambda_s(-2V^{(s)}H^{(s)T} + 2W^{(s)}H^{(s)}H^{(s)T}) \\ & + \sum_{t=1}^{n_v} \lambda_{st}(2W^{(s)} - 2W^{(t)}) + \alpha^{(s)}, \\ \frac{\partial L_1}{\partial H^{(s)}} =& \lambda_s(-2W^{(s)T}V^{(s)} + 2W^{(s)T}W^{(s)}H^{(s)}) + \beta^{(s)}. \end{aligned} \tag{4.6}$$

Using the Karush-Kuhn-Tucker (KKT) conditions that $\alpha_{ij}^{(s)}W_{ij}^{(s)} = 0$ and $\beta_{ij}^{(s)}H_{ij}^{(s)} = 0$, we have:

$$\frac{\partial L_1}{\partial W^{(s)}} \odot W^{(s)} = 0, \quad \frac{\partial L_1}{\partial H^{(s)}} \odot H^{(s)} = 0. \tag{4.7}$$

Solving the above equations, we derive the following update rules:

$$\begin{aligned} H^{(s)} &\leftarrow H^{(s)} \odot \frac{W^{(s)T}V^{(s)}}{W^{(s)T}W^{(s)}H^{(s)}}, \\ W^{(s)} &\leftarrow W^{(s)} \odot \frac{\lambda_s V^{(s)}H^{(s)T} + \sum_{t=1}^{n_v} \lambda_{st}W^{(t)}}{\lambda_s W^{(s)}H^{(s)}H^{(s)T} + \sum_{t=1}^{n_v} \lambda_{st}W^{(s)}}. \end{aligned} \tag{4.8}$$

These update rules form the solution for the pair-wise CoNMF algorithm's iterative execution. It is easy to see that $W^{(s)}$ and $H^{(s)}$ are non-negative after each update. Moreover, it is provable that the objective

function J_1 is non-increasing under the above iterative updating rules, and the convergence is guaranteed. The proof can be shown by constructing the auxiliary function similar to [115]. We provide the proof in appendix.

Normalization

While the above provides a sound solution for the optimization, in practice we find that inserting a normalization step is important. The above solution is guaranteed to minimize the objective function with local minima, but we notice that this solution does not always lead to meaningful results. There are two possible reasons for this: (1) the W matrices of the different views might not be comparable at the same scale; (2) there is a case that the value of objective function is always decreased but which does not progress towards a solution. To see the case, let us consider a solution $W^{(s)}$ and $H^{(s)}$. In the next iteration, the value of J_1 can be decreased by the update:

$$H^{(s)} \leftarrow cH^{(s)}, \quad W^{(s)} \leftarrow \frac{1}{c}W^{(s)}, \tag{4.9}$$

where c is a constant larger than 1. Under these update rules, the first term of J_1 in Eq. (4.3) (the combination of factorization of different views) remains unchanged, while the second term (co-regularization function) is decreased. In this case, J_1 is decreased through just scaling the $W^{(s)}$ and $H^{(s)}$, which is not meaningful.

We can solve both problems by normalizing the W matrices of the different views to make them comparable with each other, and effectively disallowing scaling. Notice that each column vector of $W^{(s)}$ represents a cluster, whose elements give the strength of association of the items to the cluster. As such, normalizing the column vectors of $W^{(s)}$ makes the cluster assignments of different views comparable. As our preliminary analysis (Section 4.3) has shown that the vector based L_2 norm is more effective in offsetting the influence of vector length for clustering, we adopt

the L_2 norm.

Formally, let $Q^{(s)}$ be the diagonal matrix with values $Q^{(s)}_{jj} = \sqrt{\sum_i W^{(s)2}_{ij}}$. Then the normalization strategy works as follows:

$$W^{(s)} \leftarrow W^{(s)}Q^{(s)-1}, \quad H^{(s)} \leftarrow Q^{(s)}H^{(s)}. \tag{4.10}$$

Note that $H^{(s)}$ is scaled by $Q^{(s)}$ correspondingly. In applying this simultaneous normalization, the value of the first term of Eq. (4.3) remains unchanged, while the co-regularization function is then forced to become meaningful as the coefficient matrices from different views are comparable.

With this modified procedure, we first normalize the W and H matrices of all views, and then execute the update rules during each iteration. In each iteration, the update rules decrease the value of J_1 with the normalized W and H (we term it *normalized descent*). While the normalization process may change the original value of J_1 before updating, the algorithm may not naturally converge. However, we argue that this normalized descent is more meaningful than purely decreasing the value of J_1, because it avoids both the comparable problem and scaling problem.

4.4.3 Cluster-wise CoNMF

Adopting the L_2 normalization admits another possible implementation of CoNMF. As the column vector of the coefficient matrix W represents a cluster, when we adopt the vector-based L_2 norm, each entry of W^TW gives the cosine similarity between two clusters. As such, W^TW can then be interpreted as the pair-wise cluster similarity matrix.

This leads to a natural definition for a cluster-wise paradigm of CoNMF. We define the co-regularization function of cluster-wise CoNMF as follows:

$$R_2 = \sum_{s,t} \lambda_{st} ||W^{(s)T}W^{(s)} - W^{(t)T}W^{(t)}||. \tag{4.11}$$

Following the same process of optimization as in Section 4.4.2, we obtain the following update rules for cluster-wise CoNMF:

$$
\begin{aligned}
H^{(s)} &\leftarrow H^{(s)} \odot \frac{W^{(s)T}V^{(s)}}{W^{(s)T}W^{(s)}H^{(s)}}, \\
W^{(s)} &\leftarrow W^{(s)} \odot \frac{\lambda_s V^{(s)}H^{(s)T} + 2\sum_t \lambda_{st} W^{(s)}W^{(t)T}W^{(t)}}{\lambda_s W^{(s)}H^{(s)}H^{(s)T} + 2\sum_t \lambda_{st} W^{(s)}W^{(s)T}W^{(s)}}.
\end{aligned}
\tag{4.12}
$$

Note that the update rules for $H^{(s)}$ of both CoNMF instantiations are the same, and are equivalent to standard NMF. This is because our proposed CoNMF only makes soft regularization with respect to the W matrices, while the H matrices – which represent the factorization of each individual view – remain unchanged. This desirable property effectively retains the information of each view during the factorization process. We discuss this property in Section 4.5.3.

4.4.4 Initialization

As the objective function of NMF is non-convex, the iterations only find locally-optimal solutions. Under standard NMF, W and H are initialized randomly. However, research on NMF have found that proper initialization plays an important role in the performance of NMF in many applications [12, 68]. It is reported that all NMF algorithms are sensitive to the initialization [68] and it is beneficial to perform pre-training. With multi-view clustering in mind, we propose a pre-training method to initialize CoNMF more effectively based on k-means.

Running k-means yields two outputs: the cluster assignment of each item and the centroid of each cluster. We propose to use these outputs to initialize W and H, respectively. We initialize the W matrix uniformly for all views while initializing the H matrix separately for each view. This is because the W matrices will be softly regularized with each other, while the H matrices are updated separately to represent the factorization of each

view.

Initialization of W matrices. To initialize W, we first run k-means on the combined view. The clustering assignments can be represented as a $m \times K$ cluster membership matrix M, such that $M_{ik} = 1$ if and only if item i is assigned to cluster k, otherwise $M_{ik} = 0$. As W is the coefficient matrix denoting the cluster membership, M can be used to initialize W. We propagate the $M_{ik} = 1$ entries as-is in $W^{(s)}$, but importantly, set all $M_{ik} = 0$ entries to a random number r in the range $(0, 1)$, instead of 0. This is needed to prevent the search space from becoming too sparse prematurely, as under the multiplicative CoNMF update rules, zero entries lead to a disconnected search space and result in overly localized search. The proposed initialization smooths out the initial search space, dealing with sparsity, while conforming to the same k-means combined view clustering in the first iteration.

Initialization of H matrices. For the initialization of each $H^{(s)}$, we first run k-means on the view s. Let the centroid of a cluster be a vector $\mathbf{c}_k^{(s)}$, then all centroids of the clustering can be represented as a matrix $C^{(s)} = [\mathbf{c}_1^{(s)}, ..., \mathbf{c}_K^{(s)}]^T$. We use $C^{(s)}$ as the initialization of $H^{(s)}$. The reasons are as follows. The factorization of NMF can be written as

$$V_{i\cdot} \approx \sum_{k=1}^{K} W_{ik} H_{k\cdot}, \tag{4.13}$$

where $V_{i\cdot}$ is the i-th row vector of data matrix V, $H_{k\cdot}$ is the k-th row vector of H. As such, $H_{k\cdot}$ can be seen as the basis vector to resemble the original data. In k-means clustering, each item is assigned to the cluster with nearest centroid. Therefore, the centroids of k-means clustering can also be deemed as the K basis vectors of the original data. As such, using the centroids to initialize H places them in the same space initially, which is more meaningful than random initialization. Similarly, as the update

rules of $H^{(s)}$ are multiplication-based and $C^{(s)}$ may be very sparse, which may cause shrinkage of the search space. We add a small constant ϵ to each element of $C^{(s)}$ to avoid the shrinking effect.

4.4.5 Time Complexity Analysis

We now analyse CoNMF's time complexity, using standard NMF as the basis for big O notation.

CoNMF is essentially an extension of NMF for multiple data matrices. It can be shown that the cost for NMF's update rules in each iteration is $O(nmK)$. As CoNMF's update rule for each $H^{(s)}$ is same with the original NMF, its cost is also $O(nmK)$. For each $W^{(s)}$ of pair-wise CoNMF in Eq. (4.8), the additional cost in terms of plain NMF is the second term of the numerator and denominator, whose time complexity is $O(n_v mK)$. As such, the time complexity of update rules of pair-wise CoNMF is $O(n_v mK + nmK)$. As n_v denotes the number of views, which is a small constant (in our comment-based clustering, $n_v = 3$) *s.t.* $n_v \ll n$, this yields $O(n_v mK + nmK) \approx O(nmK)$. Similarly, for cluster-wise CoNMF, the time complexity of update rules of each view is $O(n_v mK^2 + nmK) \approx O(nmK)$. Therefore, the time complexity of CoNMF update rules in each iteration is $O(n_v nmK)$, as there are n_v views to update, making CoNMF a linear extension of NMF. We empirically verified this in our experiments, as the actual running time of CoNMF was similar to running NMF on the three single views in series.

In real applications, although n may be very large, the data matrix is typically very sparse. As such, the number of actual operations can be far less. In addition, the multiplication-based update rules of our proposed CoNMF solutions further reduce the calculation, especially in later iterations. Distributed computation strategies for NMF with MapReduce [81]

Table 4.3: Per-view demographics for our datasets.

Dataset	Item #	Des.	Com.	Usr.
Last.fm	9,694	14,076	31,172	131,353
Yelp	2,624	1,779	18,067	17,068

can also be used on CoNMF, ensuring that CoNMF can be applied to large-scale data.

4.5 Experiments

In this section, we evaluate the proposed CoNMF methods for clustering items based on user comments.

4.5.1 Experimental Settings

We experiment with two datasets: Last.fm and Yelp. Table 4.3 gives summary demographics over the two datasets.

Last.fm. This dataset is the source of the preliminary study described earlier. Last.fm lists 26 music genres. We use 21 of these, which are shown in Figure 4.2. We exclude "world", "60s", "70s", "80s", "90s", which we feel are less reflective of a particular music style. For each of the 21 genres' music page, we crawl the artists tagged to it. As an artist may be tagged with multiple genres, we retain only artists tagged to a single genre, to facilitate hard clustering evaluation. For each artist, we crawl his or her bio description and user comments. In total, our Last.fm dataset consists of 9,694 artists, 455,457 users and 2,993,222 comments.

After the reduction on features described in Section 4.3, we arrive at a reduced set of 14,076 description features (unique tokens), 31,172 comment features and 131,153 unique users. The following experiments are on the reduced dataset.

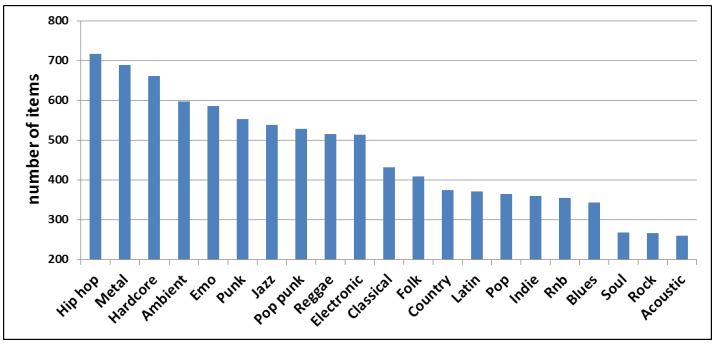


Figure 4.2: Items per category in our Last.fm dataset.

Yelp. This dataset is a subset of the Yelp Challenge Dataset (YDC)[3], including 11,537 items (businesses), 229,907 comments and 43,873 users. There are 22 first-level categories and each item is associated with relevant categories. Retaining only items that are unambiguously mapped to only one first-level category, we obtain 9,537 items. Figure 4.3 shows the statistics of number of items per category on this dataset. As the distribution is very skewed: the top category "restaurants" takes 39.9% items and the top three categories take 64.5% items. Such a skewed distribution influences the clustering evaluation greatly. To balance the number of items per category, one common way is to randomly sample some items for the large categories [83, 65]. However, this makes evaluation unstable and hard to replicate. As such, we further limit our dataset to categories with that have only items in the range of 100 to 500. Our final Yelp dataset consists of 2,624 items from 7 categories: *Health & Medical*, *Active Life*, *Local Services*, *Pets*, *Nightlife*, *Home Services* and *Arts & Entertainment*. This dataset consists of three views as well. The comment words view and users view are extracted the same way as in Last.fm; for the item-intrinsic view (description view), we use the businesses' names.

We compare with the following algorithms:

1. SVD. We run SVD on the data matrix, using the objective latent number of dimensions as K, then cluster the reduced space using k-means.

[3] `http://www.yelp.com/dataset_challenge`

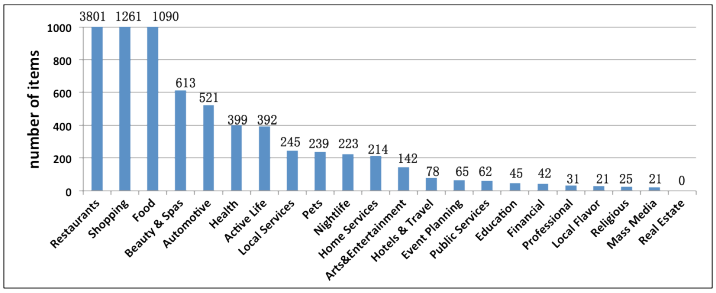


Figure 4.3: Items per category in our Yelp dataset.

This is a typical SVD workflow for clustering [135].

2. MMLDA [110]. Multi-Multinomial LDA is an extension of LDA for clustering webpages from content words and social tags, which can be seen as two views. Latent topics of words and tags are generated from the same multinomial distribution. As it is a two-view clustering algorithm, we merge the two text-based views (description and comment words view) into a single "words" view, then run the algorithm on the words view and users view, to derive the final clustering. We use the EM implementation of [27]. The topic prior is set to be 0.7, as suggested by the authors.

3. CoSC [65]. This is a co-regularization based extension of spectral clustering algorithm, designed specifically for multi-view clustering. We use the default Gaussian kernel to build the affinity matrix and set the regularization parameters to be 0.01, as suggested by the authors.

4. MultiNMF [83]. This is a consensus-based regularization solution for NMF on multi-view clustering. As the authors provide a NMF-based initialization, we use their suggested initialization method, setting the regularization parameters uniformly as 0.01 as suggested. Trying other values, we also find its performance to be consistent. Initially, MultiNMF normalizes the data matrix using L_1-whole, which has been shown to be sensitive to the vector length. For this reason, we further evaluate a solution that attempts to remove the influence of vector length. This solution, which we term, MultiNMF-L_2, first conducts item-based L_2 norm before L_1-whole,

and then runs MultiNMF.

For fair comparison, we consider all three views as equally important in our comment-based clustering. In the CoNMF settings, the regularization parameters are set to 1 for all views and datasets. We study the parameter settings in Section 4.5.4. As the W matrix of either view can be used for clustering, we report the performance of the best view[4]. For each method, 20 test runs with different random initializations were conducted and the average score is reported. We evaluate the performance of hard clustering by using clustering accuracy and F_1, as detailed in [45]. In the following, we report statistical significance (judged at the 5% level by a one-tailed two-sample t-test) where appropriate.

Limitation of Settings. We point out that there are two limitations of the evaluation setups that might introduce possible evaluation bias. First, restricted by some baselines (*e.g.,* k-means and CoSC) and the chosen evaluation metrics, we evaluated it as a hard clustering problem. Thus, we filtered out items that were labelled with more than one category. However, this operation may change the inherent regularities of data, since the original dataset has been reduced. So a more comprehensive way is to evaluate it as a soft clustering task (note that our proposed CoNMF is suitable for soft clustering). Second, due to the uneven distribution of categories, we only considered the categories of roughly equal size. Although balancing the dataset is a standard setting of previous works on clustering [110, 65], it also reduces the dataset considerably and may introduce bias. It would be interesting to see how do the clustering algorithms perform in the real-world skewed data cases.

[4]We report the performance of the best view mainly for the evaluation purpose, which tests the upper bound that can be achieved by each method. In practice, when "ground-truth" clustering is not available, we expect that the practitioner shall have the domain expertise to select the best view, which is essentially the most reliable data source.

Table 4.4: Single-view clustering results. The best performing algorithm's results are bolded.

Metric	Accuracy (%)			F_1 (%)		
View	**Des.**	**Com.**	**Usr.**	**Des.**	**Com.**	**Usr.**
Last.fm						
k-means	23.5	30.1	34.5	14.5	16.8	14.7
SVD	28.2	27.6	28.0	**24.5**	23.4	24.5
NMF	**29.5**	**39.1**	**43.6**	17.4	**28.0**	**31.6**
Yelp						
k-means	25.2	56.3	**25.0**	26.6	50.2	**26.4**
SVD	23.7	23.8	19.6	22.3	22.8	19.8
NMF	**37.2**	**60.2**	23.6	**27.5**	**57.0**	21.5

4.5.2 Single-view Clustering Evaluation

Running clustering on the single views establishes a baseline for comparison against multi-view clustering. It also allows us to compare the different single view clustering algorithms: k-means, SVD and NMF.

For Last.fm (Table 4.4, top), NMF achieves the best performance most often. The performance variation across different views is consistent in k-means and NMF: the users view performs best, and the description view performs worst. SVD, in contrast, yields consistent sub-par performance across all views, even when we vary the K for the number of latent dimensions (not shown). As SVD maps the data into orthogonal bases, which may lead to negative values, SVD's clusters are difficult to interpret naturally [135]. Thus, it is inappropriate to judge clustering credibility of the views. The results of SVD on the Yelp dataset also reflect this.

For Yelp (Table 4.4, bottom), the comment words view performs best, and the users view performs worst. Additionally, the gap between different views' performance are larger than those for Last.fm. We posit that the disparity will challenge standard multi-view clustering algorithms, as the views with poor performance may degrade the clustering of the well-performing views.

Table 4.5: Multi-view clustering results (mean ± standard deviation with 95% confidence intervals).

Dataset	**Last.fm**		**Yelp**	
Metric	**Acc. (%)**	**F_1 (%)**	**Acc. (%)**	**F_1 (%)**
k-means	40.1 ± 2.5	24.2 ± 1.9	58.2 ± 7.2	52.2 ± 6.5
SVD	29.7 ± 4.5	24.2 ± 3.1	23.0 ± 1.8	21.5 ± 2.4
NMF	45.5 ± 3.2	35.6 ± 1.9	58.5 ± 6.8	51.8 ± 5.6
MMLDA	35.2 ± 1.6	27.5 ± 1.5	48.1 ± 7.3	47.1 ± 6.8
CoSC	$\mathbf{51.7 \pm 2.3}^*$	$\mathbf{38.9 \pm 1.7}^*$	60.8 ± 2.7	56.4 ± 3.0
MulNMF	29.9 ± 1.8	21.6 ± 1.3	31.6 ± 2.4	24.2 ± 1.5
MulNMF-L_2	45.5 ± 2.3	31.7 ± 1.6	30.2 ± 2.6	24.8 ± 1.5
CoNMF-P	$\mathbf{51.9 \pm 2.5}^*$	$\mathbf{38.8 \pm 1.8}^*$	$\mathbf{67.6 \pm 4.6}^*$	$\mathbf{63.8 \pm 3.7}^*$
CoNMF-C	49.7 ± 2.5	36.2 ± 1.8	$\mathbf{67.3 \pm 5.4}^*$	$\mathbf{63.6 \pm 4.9}^*$

4.5.3 Multi-view Clustering Evaluation

Table 4.5 shows the results of multi-view clustering. K-means, SVD and NMF are run on the combined view. CoNMF-P achieves the best performance in all cases, while CoSC and CoNMF-C achieve comparable performance on Last.fm and Yelp, respectively. Although the difference between CoNMF-P and CoNMF-C is less salient for Last.fm, it is consistent and statistically significant.

We also note that the standard deviation in Yelp is generally larger than Last.fm, which we attribute to the larger performance gap in the single view clustering: the performance gap (accuracy / F_1) in terms of k-means between the comment words and users view is 31.3% / 23.8%; in contrast, the largest gap in Last.fm (between users and description views) is 11.0% / 0.2%.

Single view clustering on the combined view leads to mixed results: sometimes better and sometimes worse. SVD does not show significant improvement, k-means improves only for Last.fm, and NMF does better for Last.fm but worse for Yelp. This provides evidence that when views differ in quality, simply combining all views may not lead to improved performance.

Surprisingly, MMLDA underperforms the single view clustering of k-means and NMF. A plausible explanation is that the assumption of shared distribution to generate the latent topics of words view and users view may not hold for comment-based clustering. MMLDA was originally proposed to combine words and tags for webpage clustering. Words and tags are all text-based features, which are used to describe webpages and are still homogeneous. However in comment-based clustering, the users view and the words view are entirely different in nature: the users view reflects the users who are interested in a range of items, while the words view describe items. As such, the shared distribution constraint of MM-LDA may be too hard, and a soft constraint may perform better.

MultiNMF does not outperform the single view baselines significantly. We believe both the normalization and regularization strategies of MultiNMF may be responsible. For normalization, MultiNMF proposes to use L_1-whole, which is sensitive to vector length. As can be seen in Last.fm, the original MultiNMF does not perform well, but that applying item-based L_2 norm before L_1-whole works better. In consensus-based regularization, multiple views are regularized towards a common consensus, which may decrease performance when incorporating views with lower quality. The Yelp results provide evidence for this case: NMF on the best (worst) view yields an accuracy of 60.2% (23.6%), and the resultant MultiNMF only achieves 31.6% accuracy. The large performance gap between CoNMF and MultiNMF on Yelp supports our claim that pair-wise co-regularization suffers less from noisy views, and that the joint factorization generates a better latent space for more effective clustering.

To demonstrate the difference of two regularization schemes, we show the clustering accuracy of each single view after regularization in Table 4.6. After the consensus-based regularization of MultiNMF, each view obtains similar performance and reaches a consensus. However, the information of

Table 4.6: Effect of two regularization schemes on the clustering accuracy (%) of each single view.

Dataset	Last.fm			Yelp		
View	**Des.**	**Com.**	**Usr.**	**Des.**	**Com.**	**Usr.**
MulNMF-L_2	43.4	45.0	44.8	29.8	30.9	28.9
CoNMF-P	33.2	42.4	**51.9**	50.2	**67.6**	43.4
CoNMF-C	30.3	41.3	49.7	39.6	67.3	23.6

a view itself is lost due to the consensus constraints. In contrast, CoNMF retains the performance variance across views is similar to the original NMF (Table 4.4), while improving each view's clustering performance over NMF. It is this ability that leads to the overall improvement of CoNMF over MultiNMF as in Table 4.5.

Overall, the results demonstrate the effectiveness of CoNMF for comment-based multi-view clustering. By combining all three views in a principled way, CoNMF performs consistently better than clustering in single views as well as in the combined view. In Last.fm, CoNMF achieves a comparable performance with state-of-the-art method CoSC, and outperforms other baselines significantly. In Yelp, CoNMF performs best and achieves about 7% performance gain over the best baseline, CoSC.

4.5.4 Parameter Study

There are two sets of regularization parameters in CoNMF: λ_s for each view, and λ_{st} for each pair of views. Relative λ_s values determine each view's importance in factorization; while relative λ_{st} values determine the weight of the pair's similarity constraint in co-regularization. Relative values across λ_s and λ_{st} balance the effect of factorization and co-regularization.

By default, all parameters are set to 1. Figure 4.4 shows the performance of CoNMF-P when varying λ_{st} while holding $\lambda_s = 1$ for all views. We report only the accuracy of CoNMF-P, as F_1 figures and CoNMF-C are similarly consistent. As can be seen, for both datasets, CoNMF-P is relatively

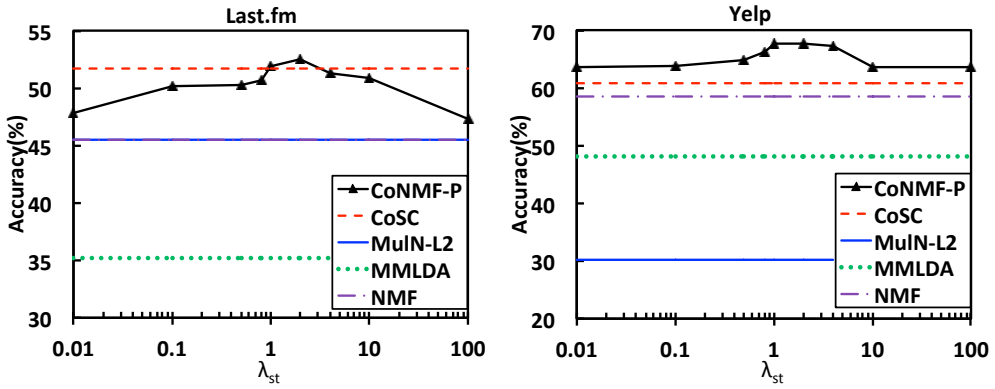


Figure 4.4: Evaluation on λ_{st} while holding $\lambda_s = 1$ for all views.

stable across a wide spectrum of settings, performing best when λ_{st} in the 1–2 range. Specifically, for Last.fm across all settings, CoNMF-P betters other baselines besides CoSC (best performance obtained when $\lambda_{st} = 2$, which is 52.5%, but is still in the same significance level with CoSC). In Yelp, over all parameter settings, the performance is significantly better than all baselines. As the three views have different clustering credibility, we also studied whether we can improve the clustering by tuning the weight λ_s of the best view. However, the performance is not improved.

These results indicate that CoNMF is stable across a wide range of parameters. As the coefficient matrices are normalized before the update rules at each iteration, they are already comparable for co-regularization. This suggest that both sets of parameters can be set to 1 when no prior knowledge informs their setting.

4.6 Discussion

We examine two specific topics worth a more detailed discussion: on the utility of the users view for comment-based clustering, and how clustering could be applied to tag generation (a topic of much current interest).

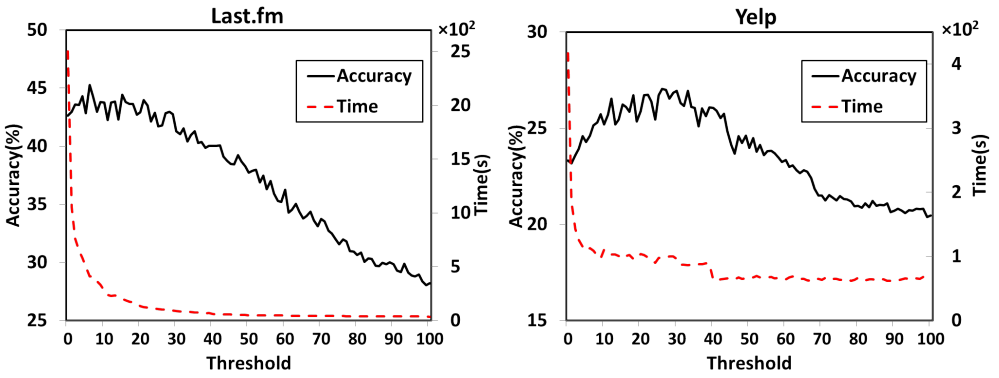


Figure 4.5: Accuracy and running time of NMF on the users view

4.6.1 Users View Utility Study

Intuitively, the utility of the users view relies on users commenting on like items, which provides evidence for clustering. The users view is most effective for users who selectively comment only many items in a single category. However, when users comment on either only one item, the value of their comment action (*n.b.*, just the action, and not the content) is zero.

We can filter users by comment frequency to try to favor the former case. We set a comment frequency threshold t, filtering out users who comment less frequently than the threshold from the original datasets. Figure 4.5 shows how the performance and running time of NMF vary with threshold t. As CoNMF extends NMF, the performance–time curve for CoNMF is consistent with NMF. We observe that a small amount of filtering is significantly useful in lessening the computational costs for NMF on the users view. As a case in point, when $t = 20$, only 2.7% and 1.4% of the original users remain in the users view of the two datasets. In such cases, the filtered users do not contribute much signal, and may even filter noise and improve performance (as seen in the Yelp dataset for $10 \leq t \leq 30$). When filtering is set too aggressively, we lose signal and accuracy drops. As a result, we conclude that a modest amount of filtering helps to boost efficiency by dropping ineffective users.

Table 4.7: Sample prominent words drawn from the clusters of the comment words view of Last.fm dataset.

Last.fm	
Cluster	**Top words**
Ambient	ambient, beauti, relax, wonder, nice, music
Blues	blue, guitar, delta, guitarist, piedmont, electr
Classical	compos, piano, concerto, symphoni, violin
Country	countri, tommi, steel, canyon, voic, singer
Hip hop	dope, hop, hip, rap, rapper, beat, flow
Jazz	jazz, smooth, sax, funk, soul, player
Pop punk	punk, pop, band, valencia, brand, untag, hi

4.6.2 Comment-based Tag Generation

In CoNMF, W is the reduced latent space of items, while H serves as the basis matrix for representing a view. As each base (row vector of H) represents a cluster, the leading elements of each base are most representative of the cluster. As the comment words view's elements correspond to comment tokens, CoNMF yields a natural method to identify representative words in the comments for each cluster. Table 4.7 and 4.8 show the words that are mapped to the leading elements in H for the comment words view from the Last.fm and Yelp dataset, respectively. For convenience, we automatically map a cluster to a category name by using the Kuhn-Munkres algorithm, shown in the "Cluster" columns. These results show that CoNMF often identifies meaningful words to represent a cluster. We also generated the top words derived from the description view (not shown), finding that the identified words are often complementary to those from comments. Our manual assessment is that the ones derived from the comments are better general descriptors for both datasets. This may be caused by the superior clustering performance of the comment words view has over the description view.

This facility of CoNMF can be utilized in downstream applications, such as tag generation. Approaches might use the top-ranked words as

Table 4.8: Sample prominent words drawn from the clusters of the comment words view of Yelp dataset.

Last.fm	
Cluster	**Top words**
Active life	class, gym, instructor, workout, studio, yoga
Arts & Enter.	golf, play, cours, park, trail, hole, theater, view
Health & Med.	dentist, dental, offic, doctor, teeth, appoint
Home services	apart, compani, unit, instal, rent, mainten
Local services	store, cleaner, cloth, dri, shirt, custom, alter
Nightlife	bar, drink, food, menu, beer, tabl, bartend
Pets	vet, dog, pet, cat, anim, groom, puppi, clinic

tags directly, or use the values in H as weights into a more sophisticated tag generation algorithm [82]. In related work, Lappas *et al.* [69] has shown that item–aspect distribution learned from social networks can improve tag generation. As the coefficient matrix resulting from CoNMF can be seen as the item–aspect distribution (after normalization via L_1 norm), we believe CoNMF's improved clustering will also lead to improved tag generation.

4.7 Conclusion

In this chapter, we investigate how to leverage user comments for clustering Web 2.0 items, an important task to several IR applications, such as search ranking, recommendation and tag generation. In an initial study on Last.fm, we confirm that two signals extracted from comments – the textual content and the commenting usernames – provide complementary information to items' intrinsic features for item categorization, and combining all three sources of information provides the best performance. Spurred by this result, we formalize this problem as a multi-view clustering problem, which aims at combining different data sources of varying quality for clustering. We first propose a general framework, CoNMF, as an extension to NMF that combine multiple views for joint factorization. Two paradigms of CoNMF – pair-wise and cluster-wise – are then introduced. Experiments

on Yelp and Last.fm datasets show that CoNMF effectively makes use of information from user comments for the clustering task.

Chapter 5

Mining Comments for Personalized Recommendation

In the two previous chapters, we demonstrated work that mined comments to improve item-centric applications in a generic manner. However, these accomplishments do not address how to tune application for any particular individual use. In this chapter, we tackle this problem. Specifically, we study how to leverage user comments to assist in generating personalized recommendation for users.

5.1 Introduction

Recommender systems serve to help users discover choice products to consume, matching users to items of potential interest (*e.g.*, products, businesses, movies, *etc.*). Among the various recommendation techniques, collaborative filtering (CF) is most widely used [143], due to its effectiveness to provide personalized recommendation based on the wisdom of crowds. For example, item-based CF technique [114] works by looking into the similar items that the target user has consumed; matrix-based CF [64] works by correlating items and users in the shared latent space. However, most ex-

isting CF techniques have focused on modeling user–item binary relations such as ratings, while neglected the "real" reasons behind the single rating score. For example, in the restaurant domain, a user may give a 5-star rating for food quality, while another user may give the same rating due to the favorable ambiance. Since existing CF techniques largely lack such fine-grained analysis, they can fail to accurately model user's interests.

Aside from ratings, most Web 2.0 systems also allow users to leave comments. As motivated by the example in Figure 1.2 (Chapter 1), these reviews often justify a user's rating, and can be implicitly thought of as componentizing the overall rating into individual comments on aspects of the rated item. This provides us new opportunities to improve collaborative recommendation by mining user comments. More specifically, by looking into the comments, we can analyze user preference in a finer granularity – *aspects*, which are the item properties that a user is interested in – thus providing more desirable recommendations to users.

In this work, we leverage user comments to address the task of personalized recommendation; that is to produce an ordered list of items that will be most appealing to a user (*cf.* top–N recommendation). Many works have addressed this task, and very recent work [8, 32, 51, 80, 92, 148] have shown the usefulness of reviews in predicting user ratings. However, the existing methods are not realistic for practical recommender systems, as: 1) they have largely focused on the rating prediction task, which is a sub-optimal fit [6, 24] to the real recommendation scenario where only a few (top) items are recommended to a user; 2) they are designed for optimizing only accuracy, while largely neglecting the transparency and explainability, crucial to users' trust of a recommender system [126]; and 3) when new data (*e.g.,* users, ratings) arrives, they need to be re-trained to adapt the new data, which is expensive and prohibitive for learning online.

To address these gaps, we propose a new recommendation method, fo-

cusing on review-aware top–N recommendation with particular attention to transparency, explainability and efficiency for online learning. Instead of the most widely used latent factor model that learns user preference in latent space, we resort to the graph-based method which models users, items and aspects (extracted from comments) as a tripartite graph. Compared to latent factor models, such graph-based methods are more transparent and explainable in generating recommendations [7, 72]. Moreover, we represent a user as her rated items and reviewed aspects, learning her preference based on the collaborative filtering and aspect filtering effects on the graph. Such a user representation makes our method easily adapt to online learning without re-training.

In the rest of this chapter, we first discuss related work in Section 5.2. Our solution first distills aspects from user comments, and then uses aspects as a supplementary data source for recommendation. We thus first discuss how to perform aspect extraction in Section 5.3, and then present our recommendation method in Section 5.4. The experiments are conducted in Section 5.5, before concluding this chapter.

5.2 Related Work

Collaborative filtering (CF) works by analyzing past interactions of users on items, and can be based on any type of user feedback, inclusive of ratings, consumption and browsing histories. Neighbor-based CF bases recommendations on similar users [13] or items [114], whereas model-based CF builds a model from user-item interactions, and leverages the abstracted model for prediction. Various model-based CF have been proposed, such as the latent factor model [143], clustering model [35], and graph model [7]. To pursue the transparency and explainability, we use the graph model as the basic recommendation model. As such this section mainly reviews

the graph-based methods for recommendation; we refer to a recent survey [88] for a more comprehensive review on the broad area of recommender system.

5.2.1 Graph-based Recommendation

Graphs form a natural representation for modeling the relationship among data objects. In recommender systems, graph models have been used widely and commercially (*e.g.,* by YouTube [7] and Twitter [40]), due to their good interpretability in generating recommendations. A typical workflow is first representing items as vertices of a graph, and then admitting recommendation as a vertex-ranking problem. For example, in YouTube video recommendation, [7] built a user–video co-view graph for video items, adopting label propagation for selecting important videos.

Besides directly working on the heterogeneous user–item graph, another family of approaches [37, 76, 84, 149] projects the user–item graph to an item–item graph (as the ranking target is an item), and then applies homogeneous graph ranking techniques, such as personalized PageRank [41]. Specifically, ItemRank [37] produced recommendations based on an item correlation graph, where entries denoted the likelihood that two items are co-rated. [84] proposed an item preference graph, where entries denoted the strength that users prefer one item over another. More recently, [149] built a tag-aware item correlation graph by considering the tag similarity among items.

We point out that a key advantage of retaining the user–item structure is that additional information can be easily incorporated by adding new types of vertices. However, existing ranking algorithms do not cater to heterogeneous structures, as they have primarily focused on homogeneous graphs [41, 150] or bipartite graphs [43]. Thus, a corresponding algorithm

must be devised to suit the specific heterogeneous graph and ranking purpose. For example, [134] modeled long-term and short-term user preference by introducing session nodes, ranking vertices by propagating user preference via *Breadth-First-Search*; [72] incorporated contexts (*e.g.,* , location, time) as vertices in the user's side, and adjusted PageRank for ranking in such a mixed bipartite graph. [15, 39] built a hypergraph to incorporate various forms of information, such as item content, tags and social information, for personalized music and tag recommendation; their ranking algorithm for hypergraph is based on the graph regularization framework [150], the same basis as our TriRank. Similar to the above works, our method retains the user–item structure, extending it to a user–item–aspect tripartite graph for modeling aspects. A key contrast in our work is that we specifically consider the ternary relationship between user, item and aspect, which has not been studied before.

5.3 Aspect Extraction

Aspect extraction, also termed as feature or attribute extraction, has a long history in review mining. The *aspects* of an item are the components and attributes of the item, which are expressed by actual words or phrases that appear in the text [145]. Early seminal work [52] proposed several language rules to extract product aspects from reviews. The basic idea is that frequent itemsets of nouns (and noun phrases) are likely to be product aspects. The rules have been widely used and extended by later work, *e.g.,* [146] considered specific phrase patterns and sentence patterns. Aside from the unsupervised rule-based methods, supervised sequence labeling techniques such as the *Conditional Random Field* have been adopted to learn aspects [57].

As our focus is to leverage aspects from user reviews, we do not con-

tribute to aspect extraction, but instead seek to maximally exploit technologies that can perform it. As such, we apply an existing state-of-the-art aspect extraction toolkit [147] that constructs a sentiment lexicon from user reviews. It creates entries that are feature–opinion word pairs with an associated sentiment polarity, represented as (F, O, S). For example, an entry such as (*service, excellent, positive*) and (*food, poor, negative*) might be extracted from a restaurant review. The key feature extraction part of the tool is a rule-based system, mainly adopting and extending the rules proposed by [52], such as identifying frequent features by association mining, pruning redundant features by using *p-support*, *etc.*. As each feature is a noun word or phrase, representing the item's property that a user comments on, we can directly use it as an aspect.

We now start the explanation of our method for personalized recommendation, starting with aspect extraction, and applied to two real-world datasets. We apply the tool with its default settings, extracting $6,025$ and $1,617$ distinct features (*i.e.,* aspects) from our datasets culled from Yelp and Amazon Electronics, respectively (datasets described later in Section 5.5). Table 5.1 shows top features extracted from the two datasets, ranked by their $tf \times idf$ score. We notice that the tool produces some features that are good but also many noisy features, such as "ive" ("I've"), "picturesmy", "l50", which are quirks of the corpus. Also, some top features are domain-specific stop words (*e.g.*, "food", "restaurant", "product"), which do not represent the specific properties of items. Despite this significant level of noise, we do not perform any post filtering on the extracted aspects to test the robustness of our proposed method to noise.

Note that in terms of manifestation, aspects and tags (in social tagging systems) look alike – they are both usually short noun phrases. However, they differ fundamentally in nature and hence utility. Tags are simple keywords that are directly annotated by users to categorize and manage

Table 5.1: Top automatically extracted aspects.

Yelp	bar, salad, menu, chicken, sauce, restaurant, rice, cheese, fries, bread, sandwich, drinks, patio
Amazon	camera, quality, sound, price, product, battery, pictures, features, screen, size, memory, lens

Table 5.2: Statistics of aspects extracted from reviews.

		User–Aspect		**Item–Aspect**	
Dataset	Aspect#	Asp.# / User	Density	Asp.# / Item	Density
Yelp	6,025	183.8	3.05%	138.0	2.29%
Amazon	1,617	61.4	3.80%	23.2	1.44%

items. Aspects, on the other hand, describe specific attributes of items, and are implicitly extracted from free-text reviews.

Table 5.2 summarizes the statistics of extracted aspects on the two datasets. We note that the densities of the user–aspect and item–aspect matrix are much higher than that of the user–item rating matrix (usually less than 1%, see Table 5.3); a good signal that the aspect matrices contain rich information useful for addressing the sparseness of the original rating matrix.

5.4 Proposed Method

We now present our proposed method for review-aware recommendation. We first introduce the tripartite graph to model the user–item–aspect ternary relation. Then we devise the generic TriRank algorithm for ranking on tripartite graphs, before deploying it for personalized recommendation. Finally, we show how to adjust TriRank for online learning and discuss several key properties including transparency, explainability and insensitivity to noisy aspects.

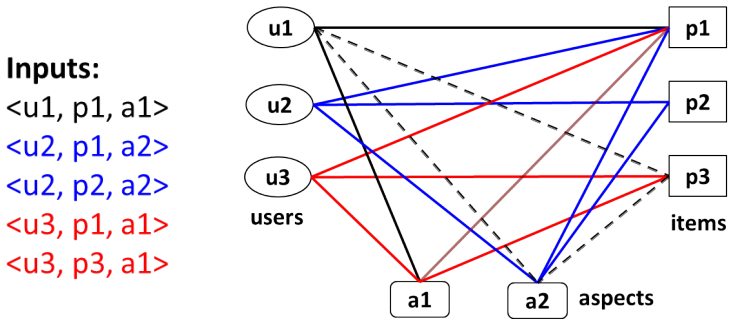


Figure 5.1: An example tripartite structure of the given inputs (the dashed line illustrates the additional input $< u_1, p_3, a_2 >$).

5.4.1 Data Model and Notation

Let $G = (U \cup P \cup A, E_{UP} \cup E_{UA} \cup E_{PA})$ be a tripartite graph, where U, P and A are vertex sets that represent users, items and aspects[1], respectively. Let E_{UP}, E_{UA} and E_{PA} be edges that represent user–item, user–aspect, item–aspect relations, respectively. Each input triple $< u_i, p_j, a_k >$ denotes that user u_i has rated item p_j with a review mentioning aspect a_k, is then represented as a triangle with edges e_{ij}, e_{ik} and e_{jk} (as in Figure 5.1). Each edge carries a weight to denote the strength of two connected vertices; edges with higher weight denote stronger more significant relations between vertices. For example, we can model user u_i's rating on item p_j as an edge with weight of e_{ij}; if u_i has never rated p_j, then the edge e_{ij} does not exist. Without loss of generality, we use the symbol R, Y and X to denote the edge weight matrix of user–item, user–aspect and item–aspect relations, respectively.

5.4.2 Tripartite Graph Ranking (TriRank)

The goal for item recommendation is to devise a ranking function $f : P \rightarrow \mathbb{R}$, which maps each item in P to a real number such that the value reflects the target user u's (predicted) preference on the item. Sorting the resultant items by score yields u's personalized item ranking. We instantiate

[1]The definition and extraction of aspects has been detailed in Section 5.3.

this method with our three matrices and thus call this method TriRank (although it can generalize to account for more than three sources). Since TriRank induces scores for all vertices in the graph, it has the important side effect of assigning scores to the aspect and user vertices: these denote u's interest on aspects and similarity with other users, respectively.

In a nutshell, TriRank assigns the ranking score of vertices by enforcing the structural smoothness and fitting constraints of the graph. By smoothness and fitting constraints, we adopt the same definitions as those common to the graph regularization framework [43, 150]:

· *Smoothness* implies local consistency: that nearby vertices should not vary too much in their scores.

· *Fitting* encodes prior belief: that the ranking function should not cause much deviation from the observations.

TriRank seeks to assign each vertex a score such that *the graph is sufficiently smooth* and *the prior belief is retained*. In the following, we first illustrate how the two constraints in the tripartite graph capture the intuition for recommendation, before describing the TriRank algorithm.

Illustrating Regularization Constraints

Let us first see how the smoothness works by considering the example in Figure 5.1. We decompose the example graph into two subgraphs in Figure 5.2 for ease of exposition. The left subfigure gives the user–item structure, where edge weights denote ratings. Assume we want to recommend items to u_1, who has only rated p_1 with a score of 5. As p_1 is connected more strongly to u_2 than u_3, u_2 is given a higher score than u_3. Finally, since the edge weights of $< u_2, p_2 >$ and $< u_3, p_3 >$ are identical, we infer that p_2 should receive a higher score than p_3. Such smoothness constraint on the user–item relation alone yields the traditional CF effect as the constraint acts to propagate the preference of similar users.

Considering aspects can provide additional evidence that influences the recommendation process. Let us continue to recommend for u_1 but base our decision on item–aspect structure (Figure 5.2(b)), where edge weight denotes the number of an item's reviews mentioning an aspect. As u_1 only previously mentions aspect a_1, enforcing smoothness would rank p_3 higher than p_2, as p_3 is more strongly connected to a_1, in contrast to p_2. This example also shows that predicting based on CF and aspect filtering yield different results; and that the *smoothness* constraint on the whole graph to combine them may be beneficial.

The fitting constraint serves as a means to personalize the ranking for each user (*cf.* shaded vertices of Figure 5.2). For a target user u, the past ratings and reviewed aspects indicate u's prior (known) preference on the vertices. It should guide the ranking process such that the resultant ranking function should be consistent with prior belief.

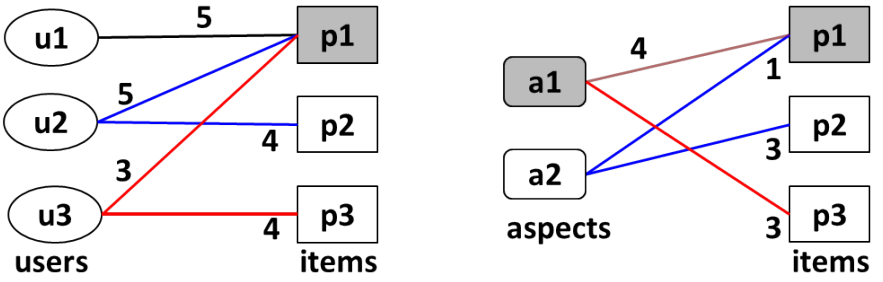
Regularization on Tripartite Graph

We now define the regularization function to implement the two constraints for ranking vertices.

Smoothness. Similar to the previous work [43] that defines a smoothness regularizer on bipartite graphs, we devise the regularizer on user–item structure as follows:

$$Q_{UP}(f) = \sum_{i=1}^{|U|} \sum_{j=1}^{|P|} r_{ij} \left(\frac{f(u_i)}{\sqrt{d_i^u}} - \frac{f(p_j)}{\sqrt{d_j^p}}\right)^2, \tag{5.1}$$

where $f(u_i)$ and $f(p_j)$ denote the final ranking scores (*i.e.,* parameters to learn); r_{ij} is the edge weight between u_i and p_j; $|U|$ and $|P|$ denote the number of users and items, respectively; d_i^u and d_j^p are the weighted degrees (sum of edge weights) of u_i and p_j, respectively, for normalization. The counterpart user–aspect and item–aspect smoothness regularizers for



(a) User-Item structure **(b) Item-Aspect structure**

Figure 5.2: Smoothness constraints on decomposed graphs from Figure 5.1. Assume $u1$ previously rated item p_1 with mentioning aspect a_1 (shaded vertices).

aspect filtering can be obtained similarly.

This smoothness regularizer can be seen as a graph kernel that measures the similarity of vertices. Although there are various kernels [118], we have purposefully chosen this one (originally introduced by [150]) due to its effective encoding of the CF effect in bipartite graph scenarios. To see this, assume we recommend for the target user u. First, minimizing Eq. (5.1) constrains a vertex's score based on its neighbors – if a user is strongly connected by many high-scoring **items** (*e.g.,* rated items of u), the user will be given a high score (*i.e.,* u assigned a score proportional to u's rated items); likewise, if an item is strongly connected by many high-scoring **users** (*i.e.,* similar users), it will have a high score. Second, the quadratic nature of the normalization suppresses the popularity of highly connected vertices; this property is essential to prevent a ranking from being dominated by popular vertices [7].

Fitting. Let the target user's prior preference on item p_j be p_j^0; then the regularizer to enforce the fitting constraint on items is defined as:

$$Q_P(f) = \sum_{j=1}^{|P|} (f(p_j) - p_j^0)^2. \tag{5.2}$$

We can similarly achieve such fitting regularizers on users and aspects.

These fitting regularizers correspond to the squared error loss that is commonly used by machine learning models in recommendation. A key difference from the rating prediction model [64, 92] that only optimizes for rated items, Eq. (5.2) also importantly takes unrated items into account (*cf.* summing over all items). This property is very desirable for the top-N task, as it aims at ranking unrated items [24].

Regularization function. To account for the heterogeneous structure of the tripartite graph, we combine the smoothness regularizer on each relation type with the fitting regularizer using different weights for each vertex type:

$$\begin{aligned} Q(f) = \alpha \sum_{i,j} r_{ij} (\frac{f(u_i)}{\sqrt{d_i^u}} - \frac{f(p_j)}{\sqrt{d_j^p}})^2 + \beta \sum_{j,k} x_{jk} (\frac{f(p_j)}{\sqrt{d_j^p}} - \frac{f(a_k)}{\sqrt{d_k^a}})^2 \\ + \gamma \sum_{i,k} y_{ik} (\frac{f(u_i)}{\sqrt{d_i^u}} - \frac{f(a_k)}{\sqrt{d_k^a}})^2 + \eta_U \sum_i (f(u_i) - u_i^0)^2 \\ + \eta_P \sum_j (f(p_j) - p_j^0)^2 + \eta_A \sum_k (f(a_k) - a_k^0)^2, \end{aligned} \tag{5.3}$$

where α, β and γ denote the weight of smoothness on user–item, item–aspect and user–aspect relation, respectively; η_U, η_P and η_A denote the weight of fitting constraint on users, items and aspects, respectively (we discuss how to set the prior preference u_i^0, p_j^0 and a_k^0 later in Section 5.4.3).

Optimizing the Regularization Function

We now minimize Eq. (5.3) to derive the final ranking scores (*i.e.,* model parameters). As the objective function is strictly convex, standard optimization techniques find a unique solution regardless of initialization. Two widely used techniques are *stochastic gradient descent* (SGD) and *alternating least squares* (ALS). SGD updates all parameters towards the negative gradients for each training instance, while ALS minimizes the objective function per parameter until a joint optimum is found (*i.e.,* coordinate-

wise descent). For this scenario, we adopt ALS over SGD as the objective function can be analytically solved for each parameter, and importantly, it does not need to set the learning rate, which is crucial to SGD's effectiveness. Additionally, it usually yields a faster convergence and is easier to parallelize than SGD.

By differentiating $Q(f)$ with respect to $f(u_i)$, $f(p_j)$ and $f(a_k)$, respectively, and letting the derivatives be 0, we obtain the iterative update rules. Let the ranking vector for items be $\vec{p} = [f(p_j)]_{|P|\times 1}$, and the prior preference vector for items be $\vec{p_0} = [p_j^0]_{|P|\times 1}$. Let similar definitions follow for $\vec{u}, \vec{u}_0$ for users, and $\vec{a}, \vec{a}_0$ for aspects. The equivalent update rules in matrix form are as follows:

$$\begin{aligned}
\vec{u} &= \frac{\alpha}{\alpha+\gamma+\eta_U} S_R \cdot \vec{p} + \frac{\gamma}{\alpha+\gamma+\eta_U} S_Y \cdot \vec{a} + \frac{\eta_U}{\alpha+\gamma+\eta_U} \vec{u}_0, \\
\vec{p} &= \frac{\alpha}{\alpha+\beta+\eta_P} S_R^T \cdot \vec{u} + \frac{\beta}{\alpha+\beta+\eta_P} S_X \cdot \vec{a} + \frac{\eta_P}{\alpha+\beta+\eta_P} \vec{p}_0, \\
\vec{a} &= \frac{\gamma}{\gamma+\beta+\eta_A} S_Y^T \cdot \vec{u} + \frac{\beta}{\gamma+\beta+\eta_A} S_X^T \cdot \vec{p} + \frac{\eta_A}{\gamma+\beta+\eta_A} \vec{a}_0,
\end{aligned} \tag{5.4}$$

where S_R is the symmetric normalized form of matrix R, defined as $[\frac{r_{ij}}{\sqrt{d_i^u}\sqrt{d_j^p}}]_{|U|\times|P|}$; S_X and S_Y are defined similarly.

5.4.3 Personalized Recommendation

Given the general TriRank algorithm, we need to cover how we obtain the initial graph (specifically, edge weights and the target user's prior preference) to concretize the generic algorithm for our review-based recommendation scenario.

Edge weights. *User–item* edge weights from relation R can be set as in traditional CF: in cases with explicit feedback, it can be the rating score[2]; for implicit feedback, whether the user has interacted with or browsed the

[2]Note that in most systems, the users are only allowed to give a score larger than 0. Thus, a score of 0 means that there does not exist an edge between the user and item, rather than a negative rating.

item (measured as either a binary yes/no, or an integer view count). Our datasets provide explicit user ratings, so we use these ratings as-is.

For the *user–aspect* relation Y and the *item–aspect* relation X, edge weights connote the degree of user interest (item speciality) with respect to the aspect. Once aspects are identified in reviews, we can use either the actual count (number of mentions) within all a user's (item's) reviews, or the review frequency (number of reviews that mention the aspects). As reviews vary in length, an aspect may occur multiple times in long reviews, but may not imply that the user pays more attention to the aspect[3]. As such, we use review frequency in our experiments. As in general IR, we take the logarithm of the review frequency, to dampen the effect of aspects that appear very frequently.

Prior preference. We need to set the prior preference vectors for the three vertex types, with respect to the target user u_i for personalization.

For *items*, the item prior preference vector $\vec{p}_0$ takes a positive value if the target user has interacted with the item, otherwise, 0. For this reason, we adopt the i^{th} row vector of R as the $\vec{p}_0$ for the target user u_i. Similarly, for *aspects*, the aspect preference vector $\vec{a}_0$ is set as the row vector of user–aspect matrix Y. As the smoothness part of Eq. (5.3) normalizes the edge weight by a vertex's degree, we also apply the L_1 norm on $\vec{p}_0$ and $\vec{a}_0$ for meaningful combination.

The user preference vector $\vec{u}_0$ should denote the target user's similarity with other users. We can set $\vec{u}_0$ based on a user's social network when it is available. In this work, we adopt the most basic approach, simply setting the target user herself as 1, and all other users as 0, so as to be consistent with $\vec{p}_0$ and $\vec{a}_0$. Note that this variable does not directly reflect user's preference on aspects or items, so its design is beyond the scope of this work.

[3]Note that this is the same argument for the analogous document frequency over collection frequency, in general IR.

Algorithm 2: TriRank for review-aware top-N recommendation.

Input: User-Item interactions R and reviews.

Offline Training (for all users):

1. Extract aspects from reviews (Section 5.3).
2. Build item–aspect matrix X and user–aspect matrix Y.
3. TF term weighting: $X = X.tf(); Y = Y.tf()$.
4. Build symmetric normalized matrices S_R, S_X, S_Y.

Online Recommendation (for target user u_i):

5. Build u_i's prior preference vectors $\vec{p}_0$, $\vec{a}_0$ and $\vec{u}_0$.
6. L_1 norm on $\vec{p}_0$, $\vec{a}_0$ and $\vec{u}_0$.
7. Iteratively run update rules Eq. (5.4), until convergence.
8. Recommend top ranked items to u_i, and explain the recommendation using top ranked aspects.

TriRank works by enforcing the collaborative filtering and aspect filtering effects, and is summarized in Algorithm 2. For convergence, one can monitor $Q(f)$'s value until it stabilizes or set a maximum number of iterations. In our experiments, TriRank converges within 20 iterations, which is sufficiently fast for online, on-demand computation. On a modest commodity desktop (Intel 3.4GHz CPU, 16GB RAM), TriRank takes 1.8 and 0.9 seconds on average to complete ranking for a target user on the Yelp and Amazon datasets, respectively.

The iterative solution Eq. (5.4) presents a more transparent view on the ranking process. The scores of items, users and aspects mutually reinforce each other – a score increase in an item will increase both the scores of the connected users and aspects; similarly, for users and aspects. The overall solution can be seen as a semi-supervised learning process [150] on graphs – with the prior preference as labeled data, the algorithm propagates the labels to other unlabeled vertices.

5.4.4 Online Learning

One strong advantage of TriRank is that it can easily support online learning (when new data streams in) without expensive re-training. In online recommendation (Algorithm 2), the main cost in TriRank lies in Step 7, *i.e.,* the iterative algorithm for refining scores. We show that this step can be achieved in constant time with suitable offline training.

Iteratively executing Eq. (5.4) can be seen as propagating information from labeled vertices, where the weights of labels are defined by $\vec{u}_0, \vec{p}_0$ and $\vec{a}_0$. As the propagation process is linear, the final result is equivalent to aggregating (*i.e.,* summing) the propagation results from each single labeled vertex. Mathematically, let M^U_{iv} be the score of vertex v propagating score from vertex u_i, *i.e.,* executing Eq. (5.4) with $\vec{u}_0$ as an one-hot vector; similar notations for M^P_{jv} and M^A_{kv}. Then the final score of vertex v (with initial preference vectors $\vec{p}_0$, $\vec{a}_0$ and $\vec{u}_0$) is given as:

$$score(v) = \sum_i \vec{u}_0(i) M^U_{iv} + \sum_j \vec{p}_0(j) M^P_{jv} + \sum_k \vec{a}_0(k) M^A_{kv}, \tag{5.5}$$

where $\vec{u}_0(i)$ denotes the i-th element of vector $\vec{u}_0$. Based on this equivalence, we can first pre-compute matrices M^U, M^P and M^A offline. The online complexity then reduces to scanning the initial preference vectors for calculating Eq. (5.5). As the initial preference vectors describe user's preference and are usually sparse, the online complexity can be seen as constant (*i.e.,* number of non-zeros in $\vec{p}_0$, $\vec{a}_0$ and $\vec{u}_0$).

Online, when a user provides new interactions (ratings) and desires updated recommendations, we update her preference vectors, and regenerate the top recommendations by Eq. 5.5. In this way, TriRank can provide instant personalization based on a user's current interactions without re-training.

User Representation

In other personalization methods such as the latent factor model, a user's history is abstracted away, represented by a unique user ID, during offline training. When new data comes, such methods are not easily adjusted to account for the new observed interactions, and the user's representation has to be rebuilt from scratch.

Instead of representing a user by an abstractive model represented by a unique ID and learning the preference affiliated with the ID, TriRank directly models a user as the collection of her rated items ($\vec{p}_0$), reviewed aspects ($\vec{a}_0$) and similar users ($\vec{u}_0$). In offline training, TriRank learns the preference associated with each single factor (*i.e.,* item, aspect, user), and personalization is achieved by the simple addition of her factors. This user representation style for recommendation, to the best of our knowledge, is novel, and makes monetization and precomputation feasible, allowing for rapid ingestion of new observations.

5.4.5 Discussion

There are three properties of TriRank that merit a more detailed discussion: explainability, insensitivity to noisy aspects, and structural ambiguity.

Explainability. As TriRank ranks items in an easily explainable way, it provides users more transparency in understanding the system behavior. We can attribute recommendations to the top-ranked aspects matching the target user and recommended item. Figure 5.3 shows a mock-up interface for explaining recommendation based on aspects, inspired by tag-based explanation [128]. Aspects are sorted by item's speciality by default, but a user can sort according to her predicted preference. This property makes the system scrutable [126], allowing a user to control how the system utilizes her reviews. For example, if a user dislikes a recommendation due to

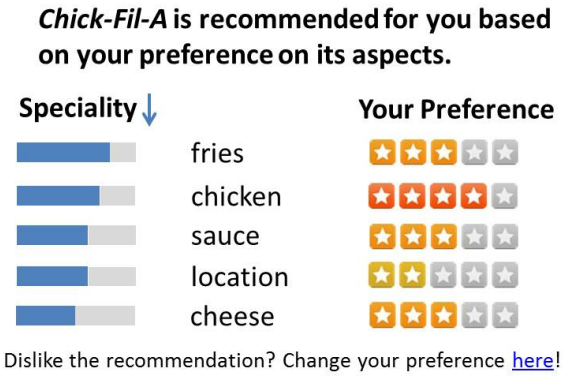


Figure 5.3: Mock user interface for showing the rationale behind recommending *Chick-Fil-A* to a user.

inaccurately-captured aspects or she has updated preference not captured in her reviews, she can edit her aspect preference. TriRank can then encode the new aspect query vector (*i.e.*, $\vec{a}_0$) and return the revised recommendations (shown later in Section 5.5.3).

This is a major advantage over the recent solutions [92, 148] which integrate reviews using a latent factor model (LFM). Such LFM methods only provide single-shot recommendation where the rationale for the recommendation is opaque. In contrast, the scrutability provided by our method easily allows recommendation to become a cyclical process – a user can iteratively interact with the recommender system, where her actions improve the system's recommendations in turn. This iterative and scrutable nature are becoming increasingly important for real-world recommender systems [126].

Insensitivity to noisy aspects. As mentioned in Section 5.3, extracted aspects are noisy. For noisy aspects which are outliers (*e.g.*, "picturemy", "150"), they usually occur less frequently in reviews as compared with those from normal aspects. As such, they will have smaller edge weights in the tripartite graph, thus exerting less impact on the ranking (see x_{jk} and y_{ik} of Eq. (5.3)). For noisy aspects which are domain-specific stop words, although they have high frequency in reviews, they actually dis-

tribute evenly for all users and items (*i.e.,* column vector of S_X and S_Y of Eq. (5.4)). As a result, they will contribute evenly across all items' ranking scores, hence not changing the relative ranking among items. As such, TriRank is relatively insensitive to noisy aspects (either outliers or stop words).

Structural ambiguity. Given a list of triples as inputs, we can uniquely represent them as a tripartite graph, but not vice versa. This is because in the tripartite graph, we cannot attribute a specific edge to an input tuple, as the conversion to the tripartite graph does not represent tuple association. More specifically, let the reviewed aspects of user u_i and item p_j be A_i and A_j, respectively. Assume u_i interacts with p_j, then the tripartite structure can not differentiate the aspects in $A_i \cap A_j$ for the interaction r_{ij}. When such ambiguities occur, they can act like unseen additional inputs, which can complement the actual observed data in a manner similar to transitive reasoning.

5.5 Experiments

We first introduce our experimental settings, and then compare its performance with other methods. We then study the utility of aspects in depth. Finally, we perform a few case studies of TriRank's recommendation output.

Datasets. We experiment with two publicly accessible datasets: Yelp[4] and Amazon Electronics [92].

1. Yelp. This is the Yelp Challenge dataset published on April 2013. It includes 11,537 items, 229,907 reviews and 45,981 users. The dataset is very sparse – 49.6% of users only made one review, making it difficult for evaluation.

[4] `http://www.yelp.com/dataset_challenge`

Table 5.3: Statistics of datasets in evaluation.

Dataset	Review#	Item#	User#	Avg	Density
Yelp	114,316	4,043	3,835	29.8	0.74%
Amazon	55,677	14,370	2,933	19.0	0.13%

"Avg" denotes the average number of reviews per user.

2. Amazon. This dataset contains user ratings and reviews on Amazon products of Electronics category, published by [92]. The original dataset contains over 800K users, 80K items and 1.3M reviews. It is more sparse than the Yelp dataset – with 77.9% of users making only one review.

Following the previous works [111, 32, 148], we filter out the items and users that have fewer than 10 reviews. Although it is a common practice in evaluating recommender algorithms, it restricts the evaluation on active users only. It real world systems, user behaviours follow a long-tail distribution, which means most users might only have very few reviews. For comprehensiveness and fairness, the recommendation performance on less active users or even cold-start users should also be evaluated. Since the focus of this work is on the utility of user reviews, it is beyond the scope of this work to concern the performance on sparse users and we leave this study as future work.

Table 5.3 summarizes the statistics of the filtered datasets. To simulate the real top-N applications, we split each dataset into three parts for training, validation and testing by time. For each user, we sort her reviews in chronological order. The first 80% are used for training, and the remaining most recent 20% are randomly split as validation set (for parameter tuning only) and test set (for evaluation).

Evaluation Metrics. Given a user, each algorithm produces a ranked list of items. To assess the ranked list with the ground-truth item set (GT), we adopt *Hit Ratio* (HR) and *Normalized Discounted Cumulative Gain*

(NDCG), which have been commonly used in top-N evaluation. The definition of the two metrics can be found in [42]. For both metrics, larger values indicate better performance. In the evaluation, we calculate both metrics for each user in the test set, and report the average score.

Baselines. We compare TriRank with the following commonly used and competitive methods in top-N evaluation:

1. **Item Popularity (ItemPop)**. Items are ranked by their popularity judged by number of ratings. Although it is not personalized, it is sometimes surprisingly competitive in top-N evaluation [24], as users tend to consume popular items.

2. **ItemKNN [114]**. This is standard item-based CF, and has been used commercially by Amazon [79] and MovieLens [128]. We adopt cosine similarity to measure the similarity among items. We test the method with different number of neighbors, finding that using all neighbors works best.

3. **PureSVD [24]**. A state-of-the-art for top-N recommendation, which performs *Singular Value Decomposition* on the whole matrix, thus directly considering all instances. unlike other latent factor methods that optimize against error only on rated instances. This property is important when applying matrix factorization models for top-N evaluation. We follow the implementation in [24], using the package SVDLIBC[5], tuning the number of latent features from 10 to 200, finding the best performance at 30.

4. **Personalized PageRank [41]**. This is a widely used graph method for top-N recommendation, *e.g.,* by [72, 134]. We perform Personalized PageRank on the user–item graph[6], and set the personalized vector same

[5] `http://tedlab.mit.edu/~dr/SVDLIBC`

[6] We also evaluated Personalized PageRank on the user–item–aspect tripartite graph. Even with optimal tuning of each edge and vertex type, performance did not improve; thus we only report Personalized PageRank's performance on the standard user–item graph.

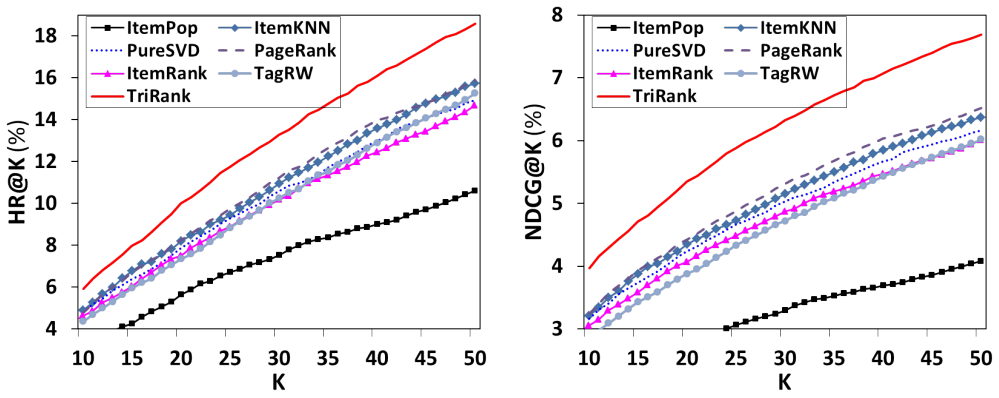
with TriRank's prior item vector $\vec{p}_0$. The damping parameter (*i.e.*, weight of the personalized vector) was respectively optimized to 0.9 and 0.3, for Yelp and Amazon datasets.

5. **ItemRank [37]**. This is another graph based method that recommends based on the item–item correlation graph. Similar to Personalized PageRank, we set the personalized vector identically as $\vec{p}_0$ and tune the damping factor.

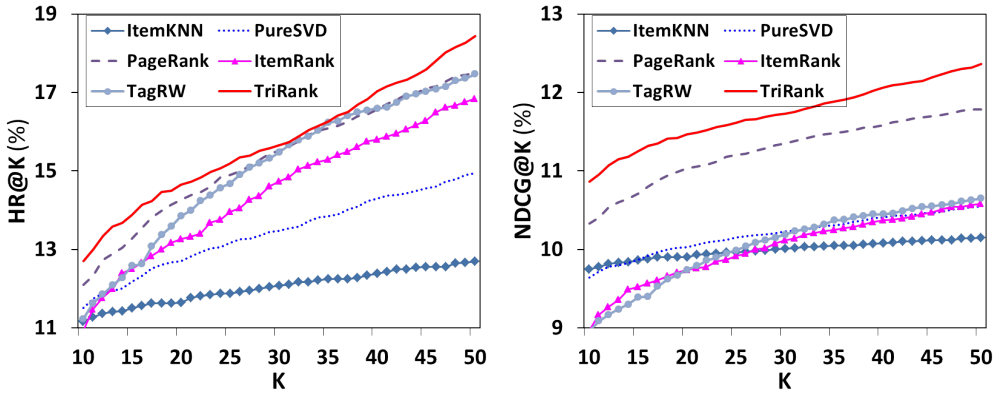
6. **TagRW [149]**. This is the state-of-the-art method to model tags for top-N item recommendation. As we have mentioned that aspects are similar with tags in terms of format, we need to compare with such a method to study how tag-aware methods perform on the task of modeling aspects. TagRW enhances ItemRank [37] by incorporating tags into building the item–item graph and performing an additional random walk on user–user graph. We feed aspects (all the same settings with TriRank) as tags into the method, and tune the five parameters of the method in a sequential way, as suggested by their paper.

As the existing review-aware methods [32, 80, 92, 148] are optimized for predicting observed ratings, it is unfair to compare with them for top-N evaluation. We validate this by evaluating the *Hidden Factors and Topics* model [92], which is state-of-the-art for review-aware rating prediction. It achieves poor top-N performance in our settings, worse than Item Popularity. Thus we do not further compare with other methods designed for rating prediction.

TriRank has six regularization parameters to tune – three for the traditional collaborative filtering effect (α, η_U, η_I) and three for the aspect filtering effect (β, γ, η_A). As performing grid-search on all six parameters simultaneously is time-consuming, we separately tune those for CF and those for aspects – first fixing β, γ, η_A as 0, searching for α, η_U, η_I; then



(a) Yelp – Hit Ratio@K (b) Yelp – NDCG@K



(c) Amazon – Hit Ratio@K (d) Amazon – NDCG@K

Figure 5.4: Performance comparison by Hit Ratio and NDCG.

performing the reverse with the optimal α, η_U, η_I. Performance was stable across many parameter settings, thus we report results for a selected set.

5.5.1 Performance Study

Figure 5.4 plots the performance when K ranges from 10 to 50. We first focus on results of the Yelp dataset. From Figure 5.4(a) and (b), we see that both metrics exhibit the same trend: TriRank performs best, outperforming all other methods with a large margin; followed by PageRank and ItemKNN, where PageRank performs slightly better than ItemKNN. PureSVD, ItemRank and TagRW obtain similar HR scores, while NDCG tells the quality of ranking: PureSVD ranks correct items higher than ItemRank and TagRW. Item Popularity performs the worst, indicating the

importance of modeling users' personalized preferences, rather than just recommending popular items.

Surprisingly, TagRW does not always outperform ItemRank, although it utilizes additional aspect information. It shows that their method for integrating tags into recommendation may not be effective for aspects. Analyzing the results, we believe that there are two reasons responsible for TagRW's inferior performance. First, they integrate aspects by transforming to the item–item and user–user similarity graph, which may lead to signal loss. Second, noisy aspects may have an adverse impact on their method, and the impact is highly dependent on the similarity measure they use. Our proposed TriRank mitigates both of these negative factors by 1) directly modeling aspects into the user–item relation as a tripartite graph, and 2) ranking vertices by regularizing the tripartite graph.

With respect to the Amazon dataset, TriRank again achieves the best performance on both metrics ($p < 0.01$ in most cases). Focusing on Figure 5.4(c) that shows the HR scores, TriRank is followed by PageRank and TagRW, which significantly outperform other methods. When K is set to 30–40, the HR differences between TriRank and PageRank and TagRW are small, but the NDCG reveals significant gaps among the three methods, indicating that TriRank successfully orders the correct items more effectively than the other two. Meanwhile, TagRW and ItemRank better PureSVD, as evaluated by HR ($K \geq 15$), but not by NDCG, which indicates the matches of TagRW and ItemRank actually occur at lower ranks. This reinforces our point that a good recall score does not necessarily translate to a high-quality ranking, hence the necessity to evaluate by ranking based measures, such as NDCG. ItemKNN performs worst among all the non-trivial personalized methods. ItemPop performed very weak, and as such, was entirely omitted in the figure to better highlight the performance of the other methods.

Looking at the interesting performance variations across the two datasets, we first notice that ItemPop only performs well on the Yelp dataset. We believe this is caused by consumption behavior differences across the two domains – people may visit popular restaurants or businesses rated in Yelp, but only purchase certain products on demand from Amazon. Similarly, ItemKNN performs strongly on the Yelp dataset (better than PureSVD), but poorly on the Amazon. One possible reason comes from data sparsity: as in Table 5.3, each item of the Amazon dataset only has 3.9 reviews on average. In such cases, the similarity measure fails in neighbor-based CF. An interesting finding is that PageRank consistently outperforms ItemRank, although both rely on Personalized PageRank with the same personalized vector. We believe the explanation is due to the fact that ItemRank ranks based on the transformed item–item correlation graph. Transforming the user–item graph to an item–item correlation graph will lose signal especially when the data is sparse, *e.g.*, when two items have no common users reviewing them. In such cases, it is more beneficial to directly rank from the user–item graph. Finally, TagRW betters ItemRank only on the Amazon dataset, indicating that the tag-based method to integrate aspects does not lead to consistent improvement. Our proposed TriRank achieves the best performance on the two datasets evaluated by both metrics, demonstrating its superiority in providing personalized item ranking to users by mining aspects in reviews.

5.5.2 Utility of Aspects

There are natural issues about aspects that we also wish to address:

1. How do the aspect-related components (*e.g.,* item–aspect and user–aspect) contribute to the performance?
2. How does the quality of aspects impact the performance? Can Tri-

Table 5.4: TriRank with different parameter settings.

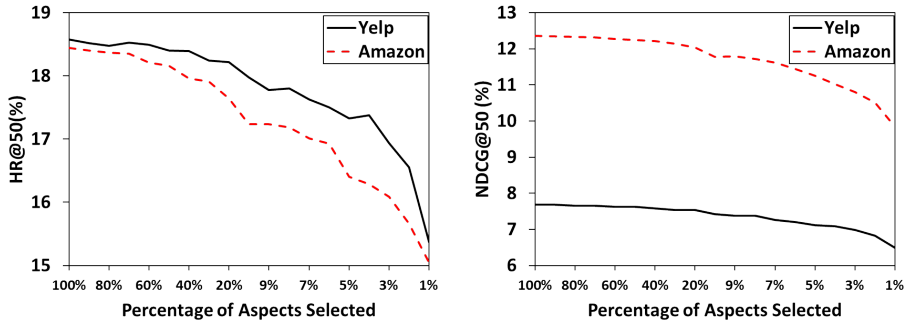
Dataset	**Yelp (@50)**		**Amazon (@50)**	
Settings	**HR**	**NDCG**	**HR**	**NDCG**
0. All set	**18.58**	**7.69**	**18.44**	**12.36**
1. $\beta = 0$ (no item–aspect)	17.05	6.91	16.23	11.31
2. $\gamma = 0$ (no user–aspect)	**18.52**	**7.68**	**18.40**	**12.36**
3. $\eta_A = 0$ (no aspect query)	18.21	7.51	17.62	12.10
4. $\beta, \gamma, \eta_A = 0$ (no aspects)	17.00	6.90	15.97	11.16
5. $\alpha = 0$ (no user–item)	11.67	4.84	10.32	5.08

Rank handle the noise in automatically extracted aspects well?

Aspect Importance Study

As TriRank is modular, with parameters for each type of vertices and edges, it is easy to answer the first question by varying the aspect-related parameters: β and γ to control the smoothness for the item–aspect and user–aspect relation, respectively, and η_A for the aspect query vector. Setting a parameter to 0 removes the corresponding effect.

Table 5.4 shows TriRank's performance with different parameter settings, evaluated at rank 50. In both datasets, when item–aspect smoothness is eliminated by setting $\beta = 0$ (Row 1), performance drops significantly. This indicates the importance of item–aspect relation, and validates our motivation that modeling user preference via decomposed aspects can yield more fidelity over modeling user–item ratings only. In contrast, when user–aspect smoothness is removed (Row 2), the performance remains unchanged. This shows that user–aspect smoothness contributes substantially less to TriRank's performance; however, we note that at least the target user's portion of the user–aspect relation can not be removed in recommendation, as the rated aspects of a user form her aspect query vector needed for recommendation. Row 3, which exhibits low performance, validates this point, as here we have removed the aspect query vector by setting η_A as 0. If we remove the modeling of aspects in its entirety (Row 4), TriRank



(a) HR vs. Aspects **(b) NDCG vs. Aspects**

Figure 5.5: TriRank *wrt.* percentage of top aspects selected.

degrades to our proposed BUIR algorithm (Chapter 3) on the user–item graph, and performs even worse. This further reveals the importance of modeling aspects for quality recommendation.

To round out our study, Row 5 shows the performance of removing user–item smoothness, which encodes standard CF in the terminology of our regularization approach. The resulting performance is worst among all settings. The user–item relation is still fundamental to model and is most important, followed by the importance of the item–aspect smoothness; user–aspect smoothness contributes least and may be removed. However, the reviewed aspects of a particular target user are still critical for capturing her personalized preference.

Aspect Quality Study

For the second question, we first rank aspects by their tf×idf score in the item–aspect matrix, and then select top scoring aspects to build the tripartite graph and inspect TriRank's performance.

Figure 5.4 shows TriRank's performance with respect to percentage of top aspects selected. As we can see, both datasets show the same trend: when the filtering ratio is moderate, performance remains largely unchanged. The filtering inflection point for both datasets is around 30%. When we filter out aspects beyond this point, performance starts to drop

significantly. This indicates that the aspects with $tf \times idf$ play a dominant role in modeling users' preference for quality recommendation. We further validate this conclusion by filtering in the reverse direction (not shown; *i.e.*, dropping the top $x\%$ aspects ranked by $tf \times idf$), finding that even a small amount of filtering (1%) leads to significant degradation. We conclude that one can safely filter out the low $tf \times idf$ scoring aspects for efficiency, as they contribute less to recommendation performance.

Interestingly, TriRank's performance does not improve when only high $tf \times idf$ aspects are utilized. Although slight improvements can be obtained with tuning, they are not statistically significant. We further evaluate TriRank with the 124 high-quality aspects selected in [148]'s work on the same Yelp Challenge dataset. Even after all parameters are re-tuned, test performance is not improved. This validates the nice property of TriRank of being relatively insensitive to noisy aspects, which are also expected to have less impact in the ranking outputs as previously explained (Section 5.4.5). TriRank can effectively utilize the merits in the automatically extracted aspects, without the need to filter out noisy aspects manually. Compared to the *Explicit Factor Model* [148] that integrates only high-quality aspects into a matrix factorization model and then generates recommendations by optimizing the predicted ratings in an opaque manner, TriRank is more transparent in leveraging aspects and also is more tolerant of low-quality aspects.

5.5.3 Case Studies

While macro-level empirical analysis are useful, it is also instructive to examine actual results to better understand the outputs of TriRank. To this end, we give two case studies drawn from the Yelp dataset to demonstrate its explainability and scrutability.

Explainability

We recap the example shown in Figure 1.2 (Chapter 1). From the first two reviews, we can see the user is interested in "chicken", although she gives low ratings to the two businesses. In the heldout test set, she reviews the business *Chick-Fil-A* with a comment "*I love Chick-Fil-A... the spicy chicken sandwhiches [sic], the lemonade, the soup, the brownies*", which further validates her preference for "chicken". As expected, TriRank ranks *Chick-Fil-A* highly (6^{th} position), mainly due to chicken being a top aspect of this business (3 of its 7 training reviews mentioned "chicken"). Examining the top items returned by PageRank, none have "chicken" as a top aspect, and most of them are popular items with more than 100 reviews. This is because random walk models are easily biased to popular items, as reported by [7]. Moreover, the third and fourth reviews show that the user is also interested in "shrimp". As a result, TriRank ranks the seafood restaurant *Red Lobster* highly in the 3^{rd} position. Although it is evaluated as a loss as the test set does not contain the item, when we checked her complete history in Yelp.com, we found she actually reviewed this restaurant later (outside of the dates in the Yelp Challenge dataset), mentioning "shrimp". Again, the recommended *Red Lobster* is not a popular item with only 7 training reviews. This case study demonstrates TriRank's capability of recommending more relevant and personalized items (not just popular items) according to a user's reviewed aspects.

Scrutability

Another key property of our TriRank instantiation is the encoding of aspect query vector $\vec{a}_0$, serving as the gateway to edit a target user's preference. We simulate the process on a sampled user[7].

[7]User ID "*omoEjYFKVV7e-DtnezeUOw*".

For this user, 9 of the 14 training reviews mentioned "service", which is the top aspect, followed by "beer". However in the test set, he reviews the business "*Total Wine & More*", whose top aspects are "wine" and "liquor". In this case, both TriRank and PageRank fail to recommend the correct item, and all top items returned do not have wine as a speciality. We simulate user feedback by editing the aspect query vector to set "wine" to a higher value, and re-run TriRank with all other parameters unchanged. In the updated ranked list, 8 of the top-10 items have "wine" as the top aspect, and the correct item "*Total Wine & More*" is ranked in 2^{nd} position.

5.6 Conclusion

In this chapter, we have studied how to utilize the aspect information in comments for top-N recommendation. We model the user–item–aspect relation as a tripartite graph, and propose TriRank, a generic algorithm for ranking the vertices of tripartite graph by regularizing the smoothness and fitting constraints. We employ TriRank for review-aware recommendation, where the ranking constraints directly model the collaborative and aspect filtering effects. To achieve personalization, we represent a user as his/her rated items, reviewed aspects and similar users, making TriRank suitable for online learning. TriRank achieves significantly improved performance over two public review datasets, even with automatically extracted aspects that have many noise. We validate TriRank as being largely insensitive to low-quality aspects, a desirable property when porting to other domains as it avoids manual efforts in filtering out noisy aspects. Most importantly, TriRank's incorporation of aspects provides users with more transparency into the recommender system behavior and affords user interaction to further improve recommendations.

Chapter 6

Conclusion and Future Work

User comments have become commonplace in Web 2.0 systems. This thesis has explored the rich evidence of user comments for three research tasks – popularity prediction, item clustering and personalized recommendation – which form the basis for many Web applications. We have proposed new methods that refine the signal from user comments for each of these tasks, and validate their generalizability and effectiveness in several cross-domain datasets.

6.1 Main Contributions

From the perspective of applications, this thesis makes the following contributions in the area of comment/review mining:

1. We utilize the temporal signal from comments for item popularity prediction, alleviating the comment sparsity problem by coupling with the social influence signal.

2. We leverage the user IDs and text features from comments to complement the clustering (categorization) of items.

3. We model user preference at a finer granularity from the underlying

aspect features to provide better personalized recommendations.

From the perspective of novel methods, this thesis makes the following contributions in applied machine learning:

4. We generalize the graph regularization semi-supervised learning framework [150] to bipartite graphs, proposing a generic algorithm BUIR for vertex ranking.

5. We extend the non-negative matrix factorization [70] technique for multi-view clustering by pair-wise co-factorization (CoNMF), and propose the pre-training and normalization methods.

6. We propose a graph-based method for recommendation that can support collaborative and content (aspect) filtering. More importantly, it can be used in online learning to provide instant personalization.

6.2 Future Work

Research on comment mining is multidisciplinary. It includes several areas such as data mining, machine learning, search and filtering, and natural language processing, among others. We believe promising future work also revolves around such multidisciplinary areas, but particularly sentiment analysis, understanding user behaviors and scalability.

In this thesis, we have focused on the comment-exploiting tasks but have forgone the use of sentiment, due to current circumstances that automated methods still do not handle the inherent difficulties in comments (*e.g.,* noise, informality, multilinguality, etc.). Recently however, a renaissance of neural network research has advanced sentiment analysis significantly, *e.g.,* Le and Mikolov [1] show paragraph vectors can achieve an accuracy over 92% on the IMDB movie reviews dataset [91]. With the advancement of research that distills the signal from raw comments – such as sentiment

analysis and structure detection – downstream comment-exploiting applications will benefit. If a similarly good accuracy can be achieved for general user comments, key hypotheses can be validated – *e.g.,* Does incorporating users' sentiments and opinions help popularity prediction and item clustering? Under what scenarios will they work? How can we effectively incorporate them into our proposed frameworks?

Another direction for future work is to expand the thesis to incorporate other user behaviors in addition to user comments, such as integrate user comments with other user behaviors, such as check-ins, votes, tags, locations, tweets and so on. Our contributions in this thesis have focused solely on mining user comments; however, to holistically understand how users fully contribute to Web 2.0, it is important to account for all of these dynamics. Some recent works have attempted to address this, for example, Yin *et al.* [138] studied the connection and difference between comments and tags in social tagging systems and Hu *et al.* [51] jointly modeled geolocations, comments and categories for rating prediction. We believe these are exemplars of the many research efforts that remain to be done along this direction.

Last but not least, scalability is a perpetual issue in data mining. To deal with user comments in real-world systems, a key challenge is in properly handling the high velocity of incoming data. New user behavior streams in, and needs to be captured and interpreted in real-time. Most existing comment-exploiting algorithms are unfortunately designed for offline processing, and key research is needed to adapt them for use in the high-velocity online settings. In our recent work [46], we have developed a new online learning algorithm for single-view matrix factorization. It will be interesting to see how to effectively extend the algorithm to support multi-view matrix factorization in an online fashion.

There are also important future work that remains to be done in the

technical domain. We highlight three open problems that are natural consequences of the techniques introduced in this thesis:

1. Optimizing parameters in the graph regularization framework (GR). In the GR framework, the hyper-parameters – that combine the smoothness and fitting constraints – are crucial for the performance. Typically, the parameters are manually tuned, which is very time-consuming and makes personalized application of GR infeasible. This significantly limits the method's effectiveness as we cannot apply it to scenarios where parameter personalization would be beneficial. For example, in our study of the TriRank method, we set the parameters uniformly for all users; however in exploratory work, we found that specific parameter settings for a subset of users improves performance, because different users might place a different degree of emphasize on personalization.

2. Extending the GR framework with ranking-based fitting constraints. Two components – smoothness and fitting constraints – form the basis of the GR framework. Fitting constraint captures the prior belief on the vertices, which should be tailored toward the target of the task. In this thesis, we have followed the traditional use of regression function as the fitting constraint. However, we point out that ranking-based functions, such as *Bayesian Personalized Ranking* [111] and *Weighted Approximated Ranking* [132], might be a better fit for ranking tasks. In the future, we will explore the ranking-oriented graph regularization method.

3. Combining the GR framework with the matrix factorization model (MF). GR models data points as a graph, and leverages its structural information for prediction. However when the inputs are sparse, the model fidelity is challenged. For example, we observed that TriRank

suffers and is unstable for sparse users with less than 5 reviews. Matrix factorization (or equivalently, latent factor models) is an effective solution to alleviate data sparsity by projecting the data into a low-dimensional latent space. It will be interesting and beneficial to combine the two complementary methods. We envision that future work in this area might reveal synergistic combination strategies that tap on the advantages of both.

Bibliography

[1] *Distributed Representations of Sentences and Documents*, 2014.

[2] M. Ahmed, S. Spagna, F. Huici, and S. Niccolini. A peek into the future: Predicting the evolution of popularity in user generated content. In *Proc. of WSDM '13*, pages 607–616, 2013.

[3] Z. Akata, C. Thurau, and C. Bauckhage. Non-negative matrix factorization in multimodality data for segmentation and label prediction. In *16th Computer Vision Winter Workshop*, 2011.

[4] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*, volume 463. ACM press New York, 1999.

[5] E. Bakshy, J. M. Hofman, W. A. Mason, and D. J. Watts. Everyone's an influencer: Quantifying influence on Twitter. In *Proc. of WSDM '11*, pages 65–74, 2011.

[6] S. Balakrishnan and S. Chopra. Collaborative ranking. In *Proc. of WSDM '12*, pages 143–152, 2012.

[7] S. Baluja, R. Seth, and D. Sivakumar. Video suggestion and discovery for Youtube: Taking random walks through the view graph. In *Proc. of WWW '08*, pages 895–904, 2008.

[8] Y. Bao, H. Fang, and J. Zhang. Topicmf: Simultaneously exploiting ratings and reviews for recommendation. In *Proc. of AAAI '14*, pages 2–8, 2014.

[9] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python*. O'reilly, 2009.

[10] M. B. Blaschko and C. H. Lampert. Correlational spectral clustering. In *Proc. of CVPR '08*, pages 1–8, 2008.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[12] C. Boutsidis and E. Gallopoulos. Svd based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition*, 41(4):1350–1362, 2008.

[13] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. of UAI '98*, pages 43–52, 1998.

[14] E. Bruno and S. Marchand-Maillet. Multiview clustering: A late fusion approach using latent models. In *Proc. of SIGIR '09*, pages 736–737, 2009.

[15] J. Bu, S. Tan, C. Chen, C. Wang, H. Wu, L. Zhang, and X. He. Music recommendation by unified hypergraph: Combining social media information and music content. In *Proc. of MM '10*, pages 391–400, 2010.

[16] C. Carpineto, S. Osiński, G. Romano, and D. Weiss. A survey of web clustering engines. *ACM Computing Surveys*, 41(3):17, 2009.

[17] Y. Cha, B. Bi, C.-C. Hsieh, and J. Cho. Incorporating popularity in topic models for social network analysis. In *Proc. of SIGIR '13*, pages 223–232, 2013.

[18] C. Chatfield. *The Analysis of Time Series: An Introduction, Sixth Edition.* Taylor & Francis, 2003.

[19] K. Chaudhuri, S. M. Kakade, K. Livescu, and K. Sridharan. Multi-view clustering via canonical correlation analysis. In *Proc. of ICML '09*, pages 129–136, 2009.

[20] S. V. Chelaru, C. Orellana-Rodriguez, and I. S. Altingovde. Can social features help learning to rank Youtube videos? In *Proc. of WISE '12*, pages 552–566, 2012.

[21] T. Chen, H. M. SalahEldeen, X. He, M.-Y. Kan, and D. Lu. Velda: Relating an image tweet's text and images. In *Prof. of AAAI '15*, pages 30–36, 2015.

[22] F. R. Chung. Spectral graph theory. 92, 1997.

[23] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation.* John Wiley & Sons, 2009.

[24] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proc. of RecSys '10*, pages 39–46, 2010.

[25] S. J. Cunningham and D. M. Nichols. How people find videos. In *Proc. of JCDL '08*, pages 201–210, 2008.

[26] O. Dalal, S. H. Sengemedu, and S. Sanyal. Multi-objective ranking of comments on web. In *Proc. WWW '12*, pages 419–428.

[27] P. Das, R. Srihari, and Y. Fu. Simultaneous joint and conditional modeling of documents tagged from two perspectives. In *Proc. of CIKM '11*, pages 1353–1362, 2011.

[28] V. R. de Sa. Spectral clustering with two views. In *ICML workshop on learning with multiple views*, 2005.

[29] J.-Y. Delort. Identifying commented passages of documents using implicit hyperlinks. In *Proc. of HYPERTEXT '06*, pages 89–98, 2006.

[30] H. Deng, M. R. Lyu, and I. King. A generalized co-hits algorithm and its application to bipartite graphs. In *Proc. of KDD '09*, pages 239–248, 2009.

[31] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine learning*, 42(1-2):143–175, 2001.

[32] Q. Diao, M. Qiu, C.-Y. Wu, A. J. Smola, J. Jiang, and C. Wang. Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars). In *Proc. of KDD '14*, pages 193–202, 2014.

[33] K. Filippova and K. B. Hall. Improved video categorization from text metadata and user comments. In *Proc. of SIGIR '11*, pages 835–842, 2011.

[34] G. Ganu, N. Elhadad, and A. Marian. Beyond the stars: Improving rating predictions using review text content. In *Proc. of WebDB '09*, 2009.

[35] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *Proc. of ICDM '05*, 2005.

[36] M. A. Gonçalves, J. M. Almeida, L. G. dos Santos, A. H. Laender, and V. Almeida. On popularity in the blogosphere. *Internet Computing, IEEE*, 14(3):42–49, 2010.

[37] M. Gori and A. Pucci. Itemrank: A random-walk based scoring algorithm for recommender engines. In *Proc. of IJCAI '07*, pages 2766–2771, 2007.

[38] D. Greene and P. Cunningham. A matrix factorization approach for integrating multiple data views. In *Proc. of ECML/PKDD '09*, pages 423–438, 2009.

[39] Z. Guan, J. Bu, Q. Mei, C. Chen, and C. Wang. Personalized tag recommendation using graph-based ranking on multi-type interrelated objects. In *Proc. of SIGIR '09*, pages 540–547, 2009.

[40] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. Wtf: The who to follow service at twitter. In *Proc. of WWW '13*, pages 505–514, 2013.

[41] T. H. Haveliwala. Topic-sensitive PageRank. In *Proc. of WWW '02*, pages 517–526, 2002.

[42] X. He, T. Chen, M.-Y. Kan, and X. Chen. Trirank: Review-aware explainable recommendation by modeling aspects. In *Proc. of CIKM '15*, pages 1661–1670, 2015.

[43] X. He, M. Gao, M.-Y. Kan, Y. Liu, and K. Sugiyama. Predicting the popularity of web 2.0 items based on user comments. In *Proc. of SIGIR '14*, pages 233–242, 2014.

[44] X. He, M. Gao, M.-Y. Kan, and D. Wang. Bipartite graph ranking with its application to popularity prediction. *Transactions on Knowledge and Data Engineering*, 2016 (under review).

[45] X. He, M.-Y. Kan, P. Xie, and X. Chen. Comment-based multi-view clustering of web 2.0 items. In *Proc. of WWW '14*, pages 771–782, 2014.

[46] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proc. of SIGIR '16*, 2016.

[47] A. Hindle, J. Shao, D. Lin, J. Lu, and R. Zhang. Clustering web video search results based on integration of multiple features. *World Wide Web*, 14(1):53–73, 2011.

[48] T. Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine learning*, 42(1-2):177–196, 2001.

[49] C.-F. Hsu, J. Caverlee, and E. Khabiri. Hierarchical comments-based clustering. In *Proc. of SAC '11*, pages 1130–1137, 2011.

[50] C.-F. Hsu, E. Khabiri, and J. Caverlee. Ranking comments on the social web. In *Proc. of CSE '09*, pages 90–97, 2009.

[51] L. Hu, A. Sun, and Y. Liu. Your neighbors affect your ratings: On geographical neighborhood influence to rating prediction. In *Proc. of SIGIR '14*, pages 345–354, 2014.

[52] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proc. of KDD '04*, pages 168–177, 2004.

[53] M. Hu, A. Sun, and E.-P. Lim. Comments-oriented blog summarization by sentence extraction. In *Proc. of CIKM '07*, pages 901–904, 2007.

[54] M. Hu, A. Sun, and E.-P. Lim. Comments-oriented document summarization: understanding documents with readers' feedback. In *Proc. of SIGIR '08*, pages 291–298, 2008.

[55] S. Jamali and H. Rangwala. Digging Digg: Comment mining, popularity prediction, and social network analysis. In *Proc. of WISM '09*, pages 32–38, 2009.

[56] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proc. of SIGIR '00*, pages 41–48, 2000.

[57] W. Jin and H. H. Ho. A novel lexicalized hmm-based learning framework for web opinion mining. In *Proc. of ICML '09*, pages 465–472, 2009.

[58] Y. Jin, M.-Y. Kan, J.-P. Ng, and X. He. Mining scientific terms and their definitions: A study of the acl anthology. In *Proc. of EMNLP '13*, pages 780–790, 2013.

[59] N. Jindal and B. Liu. Analyzing and detecting review spam. In *Proc. of ICDM '07*, pages 547–552, 2007.

[60] F. Jing, C. Wang, Y. Yao, K. Deng, L. Zhang, and W.-Y. Ma. Igroup: web image search results clustering. In *Proc. of MM '06*, pages 377–384, 2006.

[61] S. Kabbur, X. Ning, and G. Karypis. Fism: Factored item similarity models for top-n recommender systems. In *Proc. of KDD '13*, pages 659–667, 2013.

[62] A. Kaltenbrunner, V. Gomez, and V. Lopez. Description and prediction of slashdot activity. In *Proc. of LA-WEB '07*, pages 57–66, 2007.

[63] E. Khabiri, C. fang Hsu, and J. Caverlee. Analyzing and predicting community preference of socially generated metadata: A case study on comments in the digg community. In *Proc. of ICWSM '09*, 2009.

[64] Y. Koren and R. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. Springer US, 2011.

[65] A. Kumar, P. Rai, and H. D. Iii. Co-regularized multi-view spectral clustering. In *Proc. of NIPS '11*, pages 1413–1421, 2011.

[66] T. Kuzar and P. Navrat. Slovak blog clustering enhanced by mining the web comments. In *Proc. of WI-IAT '11*, pages 293–296, 2011.

[67] H. Lakkaraju and J. Ajmera. Attention prediction on social media brand pages. In *Proc. of CIKM '11*, pages 2157–2160, 2011.

[68] A. N. Langville, C. D. Meyer, R. Albright, J. Cox, and D. Duling. Initializations for the nonnegative matrix factorization. In *Proc. of KDD '06*, pages 23–26, 2006.

[69] T. Lappas, K. Punera, and T. Sarlos. Mining tags using social endorsement networks. In *Proc. of SIGIR '11*, pages 195–204, 2011.

[70] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[71] J. G. Lee, S. Moon, and Kav?Salamatian. An approach to model and predict the popularity of online contents with explanatory factors. In *Proc. of WI-IAT '10*, pages 623–630, 2010.

[72] S. Lee, S.-i. Song, M. Kahng, D. Lee, and S.-g. Lee. Random walk based entity ranking on graph for multidimensional recommendation. In *Proc. of RecSys '11*, pages 93–100, 2011.

[73] K. Lerman and T. Hogg. Using a model of social dynamics to predict popularity of news. In *Proc. of WWW '10*, pages 621–630, 2010.

[74] B. Li, S. Xu, and J. Zhang. Enhancing clustering blog documents by utilizing author/reader comments. In *Proc. of ACM-SE '07*, pages 94–99, 2007.

[75] F. Li, M. Huang, Y. Yang, and X. Zhu. Learning to identify review spam. In *Proc. of IJCAI'11*, pages 2488–2493, 2011.

[76] M. Li, B. M. Dias, I. Jarman, W. El-Deredy, and P. J. Lisboa. Grocery shopping recommendations based on basket-sensitive random walk. In *Proc. of KDD '09*, pages 1215–1224, 2009.

[77] E.-P. Lim, V.-A. Nguyen, N. Jindal, B. Liu, and H. W. Lauw. Detecting product review spammers using rating behaviors. In *Proc. of CIKM '10*, 2010.

[78] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[79] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, Jan 2003.

[80] G. Ling, M. R. Lyu, and I. King. Ratings meet reviews, a combined approach to recommend. In *Proc. of RecSys '14*, pages 105–112, 2014.

[81] C. Liu, H.-c. Yang, J. Fan, L.-W. He, and Y.-M. Wang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. In *Proc. of WWW '10*, pages 681–690, 2010.

[82] D. Liu, X.-S. Hua, L. Yang, M. Wang, and H.-J. Zhang. Tag ranking. In *Proc. of WWW '09*, pages 351–360, 2009.

[83] J. Liu, C. Wang, J. Gao, and J. Han. Multi-view clustering via joint nonnegative matrix factorization. In *Proc. of SDM '13*, pages 252–260, 2013.

[84] N. N. Liu and Q. Yang. Eigenrank: A ranking-oriented approach to collaborative filtering. In *Proc. of SIGIR '08*, pages 83–90, 2008.

[85] Y. Liu, X. Huang, A. An, and X. Yu. Modeling and predicting the helpfulness of online reviews. In *Proc. of ICDM '08*, pages 443–452, 2008.

[86] B. Long, S. Y. Philip, and Z. M. Zhang. A general model for multiple view unsupervised learning. In *Proc. of SDM '08*, pages 822–833, 2008.

[87] C. Lu, X. Chen, and E. Park. Exploit the tripartite network of social tagging for web clustering. In *Proc. of CIKM '09*, pages 1545–1548, 2009.

[88] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang. Recommender system application developments: A survey. *Decision Support Systems*, 74:12–32, 2015.

[89] Y. Lu, C. Zhai, and N. Sundaresan. Rated aspect summarization of short comments. In *Proc. of WWW '09*, pages 131–140, 2009.

[90] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *Proc. of WSDM '11*, pages 287–296, 2011.

[91] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proc. of ACL '11*, pages 142–150, 2011.

[92] J. McAuley and J. Leskovec. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proc. of RecSys'13*, pages 165–172, 2013.

[93] M. McGlohon, N. Glance, and Z. Reiter. Star quality: Aggregating reviews to rank products and merchants. In *Proc. of ICWSM '10*, pages 114–121, 2010.

[94] F. McSherry. A uniform approach to accelerated PageRank computation. In *Proc. of WWW '05*, pages 575–582, 2005.

[95] G. Mishne and N. Glance. Leave a reply: An analysis of Weblog comments. In *Third annual workshop on the Weblogging ecosystem*, 2006.

[96] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[97] A. Mukherjee, B. Liu, and N. Glance. Spotting fake reviewer groups in consumer reviews. In *Proc. of WWW '12*, pages 191–200, 2012.

[98] C.-C. Musat, Y. Liang, and B. Faltings. Recommendation using textual opinions. In *Proc. of IJCAI '13*, pages 2684–2690, 2013.

[99] A. Y. Ng, M. I. Jordan, Y. Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[100] I. Ounis, C. Macdonald, and I. Soboroff. On the trec blog track. In *Proc. of ICWSM '08*, pages 93–101, 2008.

[101] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford InfoLab, 1999.

[102] N. Pappas and A. Popescu-Belis. Sentiment analysis of user comments for one-class collaborative filtering over ted talks. In *Proc. of SIGIR '13*, pages 773–776, 2013.

[103] J. Park, T. Fukuhara, I. Ohmukai, H. Takeda, and S.-g. Lee. Web content summarization using social bookmarks: A new approach for social summarization. In *Proc. of WIDM '08*, pages 103–110, 2008.

[104] F. Pedregosa, G. Varoquaux, Gramfort, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[105] Š. Pero and T. Horváth. Opinion-driven matrix factorization for rating prediction. In *Proc. of UMAP '13*, pages 1–13. 2013.

[106] H. Pinto, J. M. Almeida, and M. A. Gonçalves. Using early view patterns to predict the popularity of youtube videos. In *Proc. of WSDM '13*, pages 365–374, 2013.

[107] M. Potthast, B. Stein, and S. Becker. Towards comment-based cross-media retrieval. In *Proc. of WWW '10*, pages 1169–1170, 2010.

[108] M. Potthast, B. Stein, F. Loose, and S. Becker. Information retrieval in the commentsphere. *ACM Trans. Intell. Syst. Technol.*, 3(4):68:1–68:21, 2012.

[109] K. Radinsky, K. Svore, S. Dumais, J. Teevan, A. Bocharov, and E. Horvitz. Modeling and predicting behavioral dynamics on the web. In *Proc. of WWW '12*, pages 599–608, 2012.

[110] D. Ramage, P. Heymann, C. D. Manning, and H. Garcia-Molina. Clustering the tagged web. In *Proc. of WSDM '09*, pages 54–63, 2009.

[111] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt. Bpr: Bayesian personalized ranking from implicit feedback. In *Proc. of UAI '09*, pages 452–461, 2009.

[112] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proc. of WSDM '10*, pages 81–90, 2010.

[113] J. San Pedro, T. Yeh, and N. Oliver. Leveraging user comments for aesthetic aware image search reranking. In *Proc. of WWW '12*, pages 439–448, 2012.

[114] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proc. of WWW '01*, pages 285–295, 2001.

[115] D. Seung and L. Lee. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562, 2001.

[116] E. Shmueli, A. Kagian, Y. Koren, and R. Lempel. Care to comment?: Recommendations for commenting on news stories. In *Proc. of WWW '12*, pages 429–438, 2012.

[117] S. Siersdorfer, S. Chelaru, W. Nejdl, and J. San Pedro. How useful are your comments?: Analyzing and predicting youtube comments and comment ratings. In *Proc. of WWW '10*, pages 891–900, 2010.

[118] A. Smola and R. Kondor. Kernels and regularization on graphs. In *Learning Theory and Kernel Machines*, volume 2777 of *LNCS*, pages 144–158. Springer, 2003.

[119] G. Szabo and B. A. Huberman. Predicting the popularity of online content. *Communications of the ACM*, 53(8):80–88, 2010.

[120] K. Takeuchi, R. Tomioka, K. Ishiguro, A. Kimura, and H. Sawada. Non-negative multiple tensor factorization. In *Proc. of ICDM '13*, pages 1199–1204, 2013.

[121] W. Tang, Z. Lu, and I. S. Dhillon. Clustering with multiple graphs. In *Proc. of ICDM '09*, pages 1016–1021, 2009.

[122] A. Tatar, M. de Amorim, S. Fdida, and P. Antoniadis. A survey on predicting the popularity of web content. *Journal of Internet Services and Applications*, 5(1), 2014.

[123] A. Tatar, J. Leguay, P. Antoniadis, A. Limbourg, M. D. de Amorim, and S. Fdida. Predicting the popularity of online articles based on user comments. In *Proc. of WIMS '11*, pages 67–75, 2011.

[124] M. Terzi, M.-A. Ferrario, and J. Whittle. Free text in user reviews: Their role in recommender systems. In *Proc. of RecSys '11*, pages 45–48, 2011.

[125] M. Terzi, M. Rowe, M.-A. Ferrario, and J. Whittle. Text-based user-knn: Measuring user similarity based on text reviews. In *Proc. of UMAP '14*, pages 195–206. 2014.

[126] N. Tintarev and J. Masthoff. Designing and evaluating explanations for recommender systems. In *Recommender Systems Handbook*, pages 479–510. Springer US, 2011.

[127] M. Tsagkias, W. Weerkamp, and M. de Rijke. Predicting the volume of comments on online news stories. In *Proc. of CIKM' 09*, pages 1765–1768, 2009.

[128] J. Vig, S. Sen, and J. Riedl. Tagsplanations: Explaining recommendations using tags. In *Proc. of IUI '09*, pages 47–56, 2009.

[129] A. Wang, T. Chen, and M.-Y. Kan. Re-tweeting from a linguistic perspective. In *Proc. of NAACL-HLT '12*, pages 46–55, 2012.

[130] G. Wang, S. Xie, B. Liu, and P. S. Yu. Review graph based online store review spammer detection. In *Proc. of ICDM '11*, pages 1242–1247, 2011.

[131] J. Wang, H. Zeng, Z. Chen, H. Lu, L. Tao, and W.-Y. Ma. Recom: reinforcement clustering of multi-type interrelated data objects. In *Proc. of SIGIR '03*, pages 274–281, 2003.

[132] J. Weston, H. Yee, and R. J. Weiss. Learning to rank recommendations with the k-order statistic loss. In *Proc. of RecSys '13*, pages 245–248, 2013.

[133] D. T. Wijaya and S. Bressan. A random walk on the red carpet: Rating movies with user reviews and PageRank. In *Proc. of CIKM '08*, pages 951–960, 2008.

[134] L. Xiang, Q. Yuan, S. Zhao, L. Chen, X. Zhang, Q. Yang, and J. Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proc. of KDD '10*, pages 723–732, 2010.

[135] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proc. of SIGIR '03*, pages 267–273, 2003.

[136] Y. Xu, W. Lam, and T. Lin. Collaborative filtering incorporating review text and co-clusters of hidden user communities and item groups. In *Proc. of CIKM '14*, pages 251–260, 2014.

[137] W. G. Yee, A. Yates, S. Liu, and O. Frieder. Are web user comments useful for search. In *Proc. of SIGIR Workshop*, pages 63–70, 2009.

[138] D. Yin, S. Guo, B. Chidlovskii, B. D. Davison, C. Archambeau, and G. Bouchard. Connecting comments and tags: improved modeling of social tagging systems. In *Proc. of WSDM '13*, pages 547–556, 2013.

[139] P. Yin, P. Luo, M. Wang, and W.-C. Lee. A straw shows which way the wind blows: Ranking potentially popular items from early votes. In *Proc. of WSDM '12*, pages 623–632, 2012.

[140] Q. Yin, S. Wu, and L. Wang. Incomplete multi-view clustering via subspace learning. In *Proc. of CIKM '15*, pages 383–392, 2015.

[141] B. Yu, M. Chen, and L. Kwok. Toward predicting popularity of social marketing messages. In *Proc. of SBP '11*, pages 317–324, 2011.

[142] H.-J. Zeng, Q.-C. He, Z. Chen, W.-Y. Ma, and J. Ma. Learning to cluster web search results. In *Proc. of SIGIR '04*, pages 210–217, 2004.

[143] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua. Discrete collaborative filtering. In *Proc. of SIGIR '16*, 2016.

[144] K. Zhang, Y. Cheng, W. Liao, and A. Choudhary. Mining millions of reviews: A technique to rank products based on importance of reviews. In *Proc. of ICEC '11*, pages 12–19, 2011.

[145] L. Zhang and B. Liu. Aspect and entity extraction for opinion mining. In *Data Mining and Knowledge Discovery for Big Data*, volume 1, pages 1–40. Springer, 2014.

[146] L. Zhang, B. Liu, S. H. Lim, and E. O'Brien. Extracting and ranking product features in opinion documents. In *Proc. of COLING '10*, pages 1462–1470, 2010.

[147] Y. Zhang, H. Zhang, M. Zhang, Y. Liu, and S. Ma. Do users rate or review?: Boost phrase-level sentiment labeling with review-level sentiment classification. In *Proc. of SIGIR '14*, pages 1027–1030, 2014.

[148] Y. Zhang, M. Zhang, Y. Zhang, Y. Liu, and S. Ma. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *Proc. of SIGIR '14*, 2014.

[149] Z. Zhang, D. D. Zeng, A. Abbasi, J. Peng, and X. Zheng. A random walk model for item recommendation in social tagging systems. *ACM Transactions on Management Information Systems*, 4(2):1–24, 2013.

[150] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *NIPS*, pages 321–328, 2004.

[151] D. Zhou and B. Schölkopf. Regularization on discrete spaces. In *Pattern Recognition*, pages 361–368. Springer, 2005.

[152] L. Zhuang, F. Jing, and X.-Y. Zhu. Movie review mining and summarization. In *Proc. of CIKM '06*, pages 43–50, 2006.

Appendix

Here, we prove the optimality of the BUIR algorithm (proposed in Chapter 3) and convergence of the CoNMF algorithm (proposed in Chapter 4).

1. Optimality Proof of BUIR

Recall that the objective function optimized by BUIR algorithm is:

$$Q(f) = \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{m} w_{ij}(\frac{f(p_j)}{\sqrt{d_j^p}} - \frac{f(u_i)}{\sqrt{d_i^u}})^2 + \alpha\sum_{j=1}^{n}(f(p_j) - p_j^0)^2 + \beta\sum_{i=1}^{m}(f(u_i) - u_i^0)^2. \tag{6.1}$$

We show the optimality of BUIR by proving the objective function $Q(f)$ is convex. In the followings, we prove the convexity of $Q(f)$ by showing its Hessian is positive semi-definite.

The second order derivative of $Q(f)$ is:

$$\frac{\partial^2 Q}{\partial p_j \partial p_j} = 1 + 2\alpha; \quad \frac{\partial^2 Q}{\partial u_i \partial u_i} = 1 + 2\beta; \quad \frac{\partial^2 Q}{\partial p_j \partial u_i} = \frac{-w_{ij}}{\sqrt{d_j^p}\sqrt{d_i^u}}. \tag{6.2}$$

Let the matrix $\mathbf{A}$ be the $(m+n) \times (m+n)$ weighted adjacency matrix of the user-item bipartite graph. Then, the Hessian of $Q(f)$ can be written as:

$$\mathbf{H} = 2\mathbf{M} + (\mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}), \tag{6.3}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{D}$ is a diagonal matrix where each entry $\mathbf{D}_{ii}$ is the weighted degree of i-th vertex (can be an item or a user). $\mathbf{M}$ is a diagonal matrix that each entry $\mathbf{M}_{ii}$ is α or β, depending on the i-th vertex denotes an item or a user. Note that the matrix $(\mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})$ is the normalized Laplacian matrix of the graph. By spectral graph theory [22], the normalized Laplacian matrix of a graph is positive semi-definite. Meanwhile, $\mathbf{M}$ is also positive semi-definite because its eigenvalues are all non-negative (eigenvalues of a diagonal matrix are its diagonal values). Finally, the addition of these two positive semi-definite matrices is also positive semi-definite, concluding that the Hessian matrix $\mathbf{H}$ positive semi-definite.

2. Convergence Proof of CoNMF

In this section, we prove that the iterative solution Eq. (4.8) provides a local-minimum solution of the pair-wise CoNMF. The convergence of the cluster-wise CoNMF solution can be proved in a similar way.

Recall that the objective function of the pair-wise CoNMF is:

$$J = \sum_{s=1}^{n_v} \lambda_s ||V^{(s)} - W^{(s)}H^{(s)}|| + \sum_{s,t} \lambda_{st} ||W^{(s)} - W^{(t)}||, s.t. \quad W^{(s)} \geq 0, H^{(s)} \geq 0. \tag{6.4}$$

where $\|\cdot\|$ denotes the Frobenius norm. The update rules are written as:

$$\begin{aligned} H^{(s)} &\leftarrow H^{(s)} \odot \frac{W^{(s)T}V^{(s)}}{W^{(s)T}W^{(s)}H^{(s)}}, \\ W^{(s)} &\leftarrow W^{(s)} \odot \frac{\lambda_s V^{(s)}H^{(s)T} + \sum_{t=1}^{n_v} \lambda_{st} W^{(t)}}{\lambda_s W^{(s)}H^{(s)}H^{(s)T} + \sum_{t=1}^{n_v} \lambda_{st} W^{(s)}}. \end{aligned} \tag{6.5}$$

In the followings, we prove the non-increasing property of update rules Eq. (6.5) using the auxiliary function method [115]. Note that there are two parts of CoNMF objective function, the NMF part (i.e., the combination of NMF in individual matrics), and the co-regularization part. It is clear that the NMF part has been already proved by [115]. As such, in this material, we focus on the co-regularization part.

To construct the auxiliary function[1], we first calculate the gradient of the objective function J. Taking gradient with respective to w, we have:

$$\nabla_w J = 2\lambda_s \left(-vH^T + wHH^T\right) + 2\sum_{t=1}^{n} \lambda_{st} \left(w - w^t\right), \tag{6.6}$$

where $w = W^{(s)}_{\alpha\cdot}$, $w^t = W^{(t)}_{\alpha\cdot}$, $v = V^{(s)}_{\alpha\cdot}$ for the α-th line of $W^{(s)}$, $W^{(t)}$, $V^{(s)}$ taken separately as row vectors.

Given certain iteration of W and H, we denote the current value of w as $\bar{w}$. The auxiliary function with respect to the row vector w is given as:

$$G_{\bar{w}}(w) = J(\bar{w}) + \nabla_w J|_{\bar{w}} \cdot (w - \bar{w})^T + \frac{1}{2}(w - \bar{w}) \cdot K_{\bar{w}} \cdot (w - \bar{w}), \tag{6.7}$$

[1]For the conditions to be satisfied by auxiliary function, please refer to [115].

where

$$K_{\bar{w}} = diag\left\{2\bar{w}_{\beta}^{-1}\left(\lambda_s \bar{w} H H^T + \sum_{t=1}^{n}\lambda_{st}\bar{w}\right)_{\beta}\right\}_{\beta=1}^{K}$$
$$= 2\ diag\left\{\bar{w}_{\beta}^{-1}\left(\lambda_s \bar{w} H H^T\right)_{\beta}\right\}_{\beta=1}^{K} + 2\sum_{t=1}^{n}\lambda_{st}\cdot I. \quad (6.8)$$

Due to the quadraticity of J with respect to w, we have:

$$J(w) = J(\bar{w}) + \nabla_w J|_{\bar{w}}\cdot(w-\bar{w})^T + \frac{1}{2}(w-\bar{w})\cdot\nabla_w^2 J|_{\bar{w}}\cdot(w-\bar{w})^T, \quad (6.9)$$

in which the Hessian can be evaluated:

$$\nabla_w^2 J|_{\bar{w}} = 2\lambda_s H H^T + 2\sum_{t=1}^{n}\lambda_{st}\cdot I. \quad (6.10)$$

Therefore, we have:

$$K_{\bar{w}} - \nabla_w^2 J|_{\bar{w}} = 2diag\left\{\bar{w}_{\beta}^{-1}\left(\lambda_s \bar{w} H H^T\right)_{\beta}\right\}_{\beta=1}^{K} - 2\lambda_s H H^T. \quad (6.11)$$

Note that the entry-wise positiveness of matrices is enforced. Therefore we can obtain that:

$$G_{\bar{w}}(w) \geq J(w), \quad (6.12)$$

which is necessary and sufficient for $G_{\cdot}(\cdot)$ to be an auxilliary function, regarding the fact that $G_{\bar{w}}(\bar{w}) = J(\bar{w})$. Based on the above argument, the renewal taken at the arg-minimum of $G_{\bar{w}}$ each time will be non-increasing, more concretely:

$$\begin{aligned}\arg\min_{w} G_{\bar{w}} &= \bar{w} - K_{\bar{w}}^{-1}\left(\nabla_w J|_{\bar{w}}\right) \\ &= \bar{w}\odot\frac{\lambda_s v H^T + \sum_{t=1}^{n}\lambda_{st}w^t}{\lambda_s \bar{w} H H^T + \sum_{t=1}^{n}\lambda_{st}\bar{w}}. \end{aligned} \quad (6.13)$$

As such, our algorithm is exactly reproduced in its row-wise form. The convergence of the algorithm is thus proved.